AN APPLICATION OF INTERACTIVE COMPUTER GRAPHICS TO THE STUDY OF INFERENTIAL STATISTICS AND THE GENERAL LINEAR MODEL

THESIS

Stephen D. Pearce, Captain, USAF

AFIT/GSM/ENC/91S-22

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY
Wright-Patterson Air Force Base, Ohio

AFIT/GSM/ENC/91S-22

AN APPLICATION OF INTERACTIVE COMPUTER GRAPHICS TO THE STUDY OF INFERENTIAL STATISTICS AND THE GENERAL LINEAR MODEL

THESIS

Stephen D. Pearce, Captain, USAF

AFIT/GSM/ENC/91S-22

The views expressed in this thesis are those of the authors and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

AFIT/GSM/ENC/91S-22

AN APPLICATION OF INTERACTIVE COMPUTER GRAPHICS TO THE STUDY OF INFERENTIAL STATISTICS AND THE GENERAL LINEAR MODEL

Presented to the Faculty of the

School of Systems and Logistics

of the Air Force Institute of Technology

Air University

In Partial Fulfillment of the

Requirements for the Degree of

Master of Science in Systems Management

Stephen D. Pearce, B.S.

Captain, USAF

September 1991

Acknowledgements

The research conducted in this thesis was based on ideas generated primarily by Professor Dan Reynolds of AFIT and Mr. Richard Lamb. For their support and ideas I thank them. Without their constant enthusiasm for the effort completed herein, this research could not have evolved to its current level.

I would also like to thank my loving wife, Melissa, for her understanding through all of those long days, afternoons, and late nights at the computer while I performed this research and the other AFIT class work.

Table of Contents

Page

List of Figures

Page

List of Tables

Abstract

This research created a learning environment, known as the Pearce Projection Modeling Environment (PPME) which is used as a tool by a teacher and student. The PPME was developed in an effort to create a new approach to the study of the General Linear Model through a constructive and projective, geometric approach. While the geometric approach to the GLM was developed over the past century, it has not been used extensively because of the inherent complexities associated with visualizing vector spaces. With the PPME, visualization is accomplished effortlessly. The PPME is a computer program that allows the student to enter response vectors and other vectors and data associated with the GLM and observe the relationships of those vectors interactively and in three-dimensions. The PPME encourages learning through constructive development by allowing the student to modify the vectors and observe the results of his actions.

To validate the PPME as a learning tool, several data sets were generated and used to study three scenarios: The Sample Mean and Variance, A General Linear Test of the Population Mean, and An Ordinary Least Squares Simple Linear Regression.

AN APPLICATION OF INTERACTIVE COMPUTER GRAPHICS TO THE STUDY OF INFERENTIAL STATISTICS AND THE GENERAL LINEAR MODEL

I. Introduction

General Issue

The mathematics education level of students in the United States has declined to crisis levels in recent years. According to Newsweek, "one international study after another places U.S. school kids near the bottom of the heap in mathematical achievement" and suggests "*average* Japanese 12th graders have a better command of mathematics than the top 5 percent of their American counterparts." Clearly something must be done to upgrade our competency in teaching the mathematical sciences. (2:52)

This research will show how the technology of computers can be blended with techniques developed over the past century in statistics education to increase students' mathematical competency in graduate level statistical courses. One of the primary areas of statistical study at the Air Force Institute of Technology (AFIT) is the General Linear Model (GLM),

which is the main focus of two courses in Statistics. The GLM is also at the heart of this research.

Traditionally, the GLM is taught from an algebraic standpoint. This approach limits one's ability to conceptualize the true essence of the GLM. Students are encouraged to view modeling as a purely computational exercise resulting in rote memorization of the critical concepts associated with the GLM. As a consequence, students are inclined to work problems without any true understanding of what they are doing.

David Herr's research has shown how mathematicians employ the Geometric (Projective) Approach to facilitate a meaningful explanation of the GLM. Herr characterizes such an approach to the study of the GLM as "beautifully elegant [if you see it]." (6:45) Peter Bryant describes how geometric images can provide a natural way to look at the basic conceptual entities of statistics such as the mean and sample variance, and then lays a foundation for a pedagogy that uses geometry to motivate the higher and more inclusive concepts of the GLM. (1:43-46)

While Herr reviews historical efforts to justify a projective approach toward the study of the GLM and Bryant lays the mathematical cornerstones for an appropriate pedagogy, it took the thesis work of Captain Stone Hansard (AFIT-GSM-90S) to create and implement a computer-based methodology that provided a sufficient basis for the graphically-intensive

pedagogy developed by this thesis effort. Hansard demonstrated that computers could be used to help reverse the dominant transfer-leaning protocol to one that encourages students to experiment with the theory before they construct a concept base that captures the essence of the theory.

Ernst von Glaserfeld proposes that the teacher's job is not to transfer knowledge to the student. Rather, he suggests it is best to create an environment in which the student employs his intuition, and experiences success in the problem solving arena, BEFORE he is formally introduced to mathematical theory. Glaserfeld, commenting on the evolution of such a pedagogy, says,

> A conceptual model of the formation of the structures and the operations that constitute mathematical competence is essential because it, alone, could indicate the direction in which the student is to be guided. The kind of analysis, however, that would yield a step by step path for the construction of mathematical concepts has barely been begun. It is in this area that, in my view, research could make advances that would immediately benefit education practice. (4:16)

It is just such a constructive learning approach that this research attempts to implement via a system of interactive graphic computer programs designed to assist the student in learning the General Linear Model through heuristic exploration of underlying principles of the GLM from a geometric point of view. It is believed such active and visual involvement with the GLM will foster a clear visualization of its mathematical elegance and lead to a rigorous mastery of the theory of the GLM.

Specific Problem

To help close the gap between theory and competent practice of general linear modeling, a systematic process for experimenting with applications of the GLM needs to be developed. This system should portray both the elegance and mathematical rigor of the GLM in a manner any serious student can comprehend.

The student is introduced graphically to the General Linear Model in three steps. First he can study the geometric structure of the sample mean and sample variance. Next he is encouraged to conduct a hypothesis test of the population mean visually. Finally he is encouraged to experiment with the geometric structures representing a simple linear regression. Details concerning the program flow and elements used to create a graphic environment for experimenting with the GLM follow in Chapter 3.

Programs, alone, cannot satisfy the requirements for facilitating a constructive learning process. In order for the student to gain understanding of the GLM, competent guidance must be available to him. The instructor and learning protocols provide guidance for governing any educational event which uses the computer as a tool in the exploration of the General Linear Model.

Research Hypotheses

Three research hypotheses provided a framework for the development and evaluation of this computer assisted learning system.

1. The learning protocol developed by Hansard known as the VVAM (Visualization, Verbalization, Algorithmization and Mathematization) can be used to orchestrate an experiment-based environment for constructively learning and applying the concepts of the General Linear Model.

2. The projective approach toward the study of the Sample Mean, Sample Variance, and inferences using the General Linear Model can be represented and implemented graphically on an IBM compatible computer.

3. Through the use of the VVAM protocol and relevant heuristics of constructive learning, students will be able to gain an understanding and experiential appreciation of the concepts associated with general linear modeling.

Justification of Research

The basic premise of constructive or discovery learning is that the student learns primarily through experimentation. Glaserfeld supports this assumption when he says "although one can point the way with words and symbols, it is the student who has to do the conceptualizing and the operating." Although learning is ultimately the student's responsibility, the teacher's duty is "to help and guide the student in the conceptual organization of certain areas of experience." (4:16) The thrust of the

research and the research hypotheses proposed for this thesis respond directly to this requirement for a meaningful and experiential learning of mathematics.

Joseph Scandura reiterates the argument for discovery learning after reviewing the results of several experiments comparing it to "regular learning" when he says:

> The discovery group not only performed better than the expository group on tests designed to measure the transfer of heuristics but they better retained the material that had been originally taught. (11:119)

While discussing the geometric approach to statistics, D. J. Saville and G. R. Wood also agree that

> In fact, there appears to be a real need for a teaching method that bridges the gap between the two extremes: a method that conveys an understanding of the underlying mathematical principles at an elementary level. (10:205)

Clearly, constructive learning has a place in the study of mathematics.

Scope of Research

This research focuses on the development of computer programs to implement a constructive approach to learning the General Linear Model and the evaluation of the system's capacity to facilitate a student's ability to experiment geometrically with various mathematical concepts and properties of the GLM.

II. Background

The development of the geometric approach to the General Linear Model is not a new idea. According to David Herr, one of the earliest published accounts of the model was written in 1915 by Ronald A. Fisher. Fisher's use of geometry to discuss the distribution of correlation coefficients was elegant, but it was difficult to discern from his written papers. (6:44-45)

In 1933-34, M. S. Bartlett combined the use of algebra and geometry in his description of sample sizes as being vectors of dimension *n*. This allowed the reader to follow Bartlett's progressions using as much, or as little, geometry as the reader could competently engage. This meant the reader could fall back on the algebra, as required, and led to increased insight to the geometric approach. (6:45)

J. Durbin and M. G. Kendall studied the geometric approach to estimation in 1961. Their findings were similar to Fisher's in that they involved little or no algebra. While this type of approach is sometimes more elegant, it can leave the reader lost if he does not "see" the developments. (6:45)

William Kruskal used the geometric approach in the development of the Gauss-Markov estimation methods. He thought the geometric approach

provided for a more elegant and general approach. Since the geometric approach was understood by many statisticians,

> Kruskal hoped his paper would encourage more statisticians to adopt this approach to linear models. It does not appear that this hope was realized during the next 10 years or so. (6:45)

As Kruskal says, the geometric approach is available *but the mechanisms needed in order to make it an effective teaching tool just do not exist.* (6:46)

In 1967 G. S. Watson combined the algebraic and geometric approaches in order to motivate a clearer understanding of the underlying principles of least squares regression. Like Bartlett's paper, this allowed readers to revert to the common ground of algebra if needed. (6:46)

Kruskal uses the geometric approach again and uses the coordinate-free development of the linear model in a comparison of earlier works completed by Watson and Zyskind. Herr says, "the simplicity and beauty of the coordinate-free approach is clearly demonstrated by such a comparison" (6:46).

After analyzing the developments in this field, Herr describes several theories that explain why the geometric development of the General Linear Model is not used more widely. One theory is that while "the pure geometric approach convinced two generations of statisticians that geometry might be all right for a gifted few, it would never do for the masses" (6:46). Another theory proposes

> To fully appreciate the analytic geometric approach and to be able to use it effectively in research, teaching, and consulting requires that the statistician have an affinity for and talent in abstract thought. (6:46)

The common thread between the two theories is that understanding the geometric approach requires considerable additional effort for the ordinary student. While Herr discusses the historical development of the geometric approach to the General Linear Model, Peter Bryant provides a more teachable approach to the subject. (6:46)

Bryant discusses the geometry common to Statistics and Probability and how a geometric approach can lead to a more unified understanding of the concepts in each of these fields. (1:38)

"Perhaps one reason for this lack of unity is that the relevant material has not been published in the appropriate elementary-level literature," (1:38) he states. Bryant believes that if the student can understand the basic concepts that are developed through the geometric approach, then there are only "variations on a common theme," (1:38) any one of which can then be understood relatively easily. (1:38)

Bryant considers least squares regression an ideal arena in which a geometric approach can be employed. Through geometry the students can obtain a visual confirmation of exactly how the best estimate of regression parameters is obtained. Confirmation secured in two or three dimensions can

easily be extended to a subspace of n dimensions once the basic concepts are mastered in two or three dimensions. (1:40)

Bryant then discusses the orthogonal projections of a given vector onto a plane and demonstrates how the angle between the vector and plane can also be calculated through the use of inner products. The concept of the inner product can then be applied to the least squares regression where the angle can be used to determine if a good fit exists. Thus the mean of the data can be obtained geometrically. Geometrically, the error, which must be examined in order to determine how good the fit is, is the difference between the data and the mean vector. Using the Pythagorean theorem, the magnitude of the error can be calculated. Statistically this is referred to as the sum of squares of error or SSE. Sample variance is calculated by dividing SSE by its degrees of freedom. (1:41-42)

Graphically, the degrees of freedom represent the number of dimensions along which a vector is allowed to vary. Since the error vector must be orthogonal to the original vector of data, its degrees of freedom are based on the number of samples of data less one. The error vector must be orthogonal because it represents the shortest distance between the mean of the vector and the vector of data. (1:45)

The geometric approach is used with the General Linear Model in order to see how one set of data correlates with another. Graphically we can

observe this by comparing the difference between the vector of data and the projection of the mean along the model space. The inner product created by the different vectors of two or more sets of data allows us to assess the relation of the data. (1:45-46)

Bryant's developments allow the student to understand more of mathematics by allowing him to visualize what is happening. An algebraic approach often can result in a cookbook approach to statistics. Application of Bryant's methods are discussed in more detail in the analysis section of this research found in Chapter 4.

Saville and Wood also discuss how they used the geometric approach to the General Linear Model to teach two courses in statistics in which

> the aim was to introduce students to the theory and methods of analysis of variance and regression of a rigorous but elementary geometric setting, at the same time highlighting the unity of the area. (10:205)

First they present a basic overview comparing algebraic concepts such as the vector and show the geometric analogy. They continue to show how vectors are added, and how vectors are projected onto subspaces. After laying a basic geometric foundation, they demonstrate an example and explain its significance geometrically. The example is based on the reduced General Linear Model with $y_i = \mu + e_i$, where $e_i = y_i - \bar{y}$. Then y is projected onto the Model and this "best fit" is $\bar{\mu}$. By observing the vectors graphically,

the student can clearly visualize the orthogonality of the model space projection and the error vector which, when summed, form the observation vector. Other examples of the GLM are also described by the authors. (10:205-213)

In 1979 Marvin Margolis presented an article in which "the emphasis is on geometric thinking as a means of visualizing and thereby improving an understanding of methods of data analysis" (8:131). He develops a projection transformation $P=X(X^TX)^{-1}X^T$ (where X is the model space) which is used to obtain the projection of the observation matrix onto the estimation space. Using this projection transformation he develops the mean of the observation matrix. He then uses the perpendicular projection transformation (I-P) to calculate the error vector and by dividing the magnitude of the error vector by the degrees of freedom less one, the sample standard deviation can be found. (8:132-133)

This research is largely based on applications developed by Saville, Wood, and Margolis who have, as university instructors, used the geometric approach to the General Linear Model in classrooms with much success. This curriculum is but one of four commonplaces of education addressed by Gowin and Novak. In order for a student to learn effectively, we must consider the other three commonplaces: the teacher, the learner and the

governance (9:6). The commonplaces which are most accessible to improvement are the teacher, learner, and curriculum (5:9).

To facilitate the interaction between all of these commonplaces, Hansard developed a protocol that encourages a dialogue between the student and instructor concerning the key elements of curriculum through a series of learning heuristics known as the VVAM. This protocol was specifically developed with the computer in mind as the preferred tool to enhance the student/teacher interaction throughout the learning process. The first step was for the student to *visualize* the mathematical activity operationally. At this point the computer guided the student through an example step by step based on inputs from the student. The student was allowed to repeat these mathematical operations varying the inputs until he felt he could verbally express the logic of the mathematical actions under study. (5:23-24)

The next step was for the student to *verbalize* the steps of the process to the instructor. This verified to the instructor that the student understood what the steps of the process were. (5:25)

After the instructor felt that the student understood the process, the student was asked to *algorithmize* the mathematical operations. The algorithmization was different from the verbalization in that the student would write down the steps of the process in a sort of pseudo code. In order to assure an accurate pseudo code, Hansard suggests,

> An excellent way to verify the accuracy of the algorithm is by stepping through the algorithm and the program at the same time to see if both yield the same results at every step. (5:26)

The final step in the protocol requires the student to *mathematize* the process. At this point the student should review his algorithms and convert them to mathematical form using the appropriate mathematical notation. (5:26)

Together these four steps constitute the VVAM protocol. This protocol is particularly applicable whenever a student and teacher are using computers to facilitate the learning process. As a result of Hansard's verification of the VVAM protocol's efficacy, it was selected as the learning heuristic of choice for this thesis.

According to Herr, one of the major problems with the geometric approach to the GLM lies in dealing with the abstraction. This can be difficult if the student doesn't quite understand the steps. By using the VVAM, the computer becomes a tool with which the student can manipulate the subspace and gain greater insight into the General Linear Model.

The goal of this research is to combine the previous developments in the projective geometry of the General Linear Model with the VVAM learning protocol. The details explaining how this will be accomplished are included in the next chapter.

III. Methodology

Overview

Chapter 2 clearly demonstrates that geometric methods have been used by leading statisticians throughout the past century to rigorously develop the General Linear Model. Previous researchers seem to agree that a major challenge, involved in sharing this geometric development with more than a few gifted students, is the visualization of key concepts of the GLM. Although several scholars have developed different pedagogies to teach the geometric approach to the GLM, they all agree on one thing: there is a need for an interactive learning environment in which teachers and students learning the GLM would be assisted by a competent (and ideally computer-facilitated) visualization of GLM concepts. This research effort developed such an environment.

Employing the VVAM learning protocol developed by Captain Stone Hansard to govern the educating event, the system created by this thesis guides the teacher and student in their mutual discovery and mastery of a projective approach to the study of the GLM.

This chapter addresses the methodology that was used to confirm the three research hypotheses proposed in Chapter 1.

Why Use the VVAM?

Before an educational tool can be used effectively, its impact on each of the four commonplaces of educating (Governance, Curriculum, Teacher, Student) must be considered and addressed. As outlined in Chapter 2, such an environment should directly address the needs and interplay of three of the commonplaces of any educating event: the teacher, the student, and the subject matter. Since the fourth commonplace, governance, controls the manner in which the teacher and student discuss the subject under study, it is important that governance be given full and competent recognition during any educating event. Employment of the VVAM ensures this takes place in the environment created by the PPME.

According to Glaserfeld, one traditional approach to teaching mathematics is based on the assumption that instructors should teach any subject by pouring knowledge into the student. Unfortunately, complex mathematical concepts such as those involved with the GLM cannot simply be transferred. In fact, when the "transfer paradigm" is implemented, students usually end up regurgitating course material in order to pass exams. Little or no meaningful learning can take place. (4:16)

The VVAM reverses this traditional role of the teacher by allowing the student to work through problems actively before the teacher presents the theory motivating the problem-solving techniques. One of the reasons the

VVAM protocol was chosen as the learning protocol in this thesis was because it specifically emphasizes visualization during the study of mathematics. Since the most common barrier to implementing a geometric approach to the GLM is a lack of ability to orchestrate competent visualization, it seemed logical to employ a learning protocol that emphasized such "seeing." The use of the computer to facilitate visualization enlivens the curriculum. Through visualization, the teacher and student can experiment with the concepts of the GLM and receive extraordinary encouragement to discuss their mutual levels of understanding.

This exchange, or verbalization of mutual understanding between the student and teacher, is the second requirement of the four step VVAM protocol. It is the experience of this researcher that, until a student can clearly articulate his understanding of a concept, his mastery of the subject matter should not be presumed by the teacher, or himself.

There is unanimous agreement that once a student understands a concept of the GLM, he needs to demonstrate his ability to apply it. Hence, the importance of the third step of the VVAM: algorithmization, is clear. The ability of a student to construct an algorithm in pseudocode as a means of demonstrating his understanding of a concept facilitates discussions between the student and teacher concerning the steps involved in solving a mathematical problem. Once the student has verified that his pseudocode

works, the teacher can be assured that the student understands both the concept and its application. (5:25-26)

The final step of the VVAM is mathematization. This step requires the student to translate his algorithm into the formal language of mathematics using appropriate mathematical notation and operators. Competent employment of matrix algebra is particularly important at this stage. The MathCAD[1] software package provides a convenient medium for assisting the student in meeting this final requirement of the VVAM. (5:26)

Because the VVAM approach emphasizes interaction between teacher and student within the context provided by a competent visualization of relevant mathematical concepts, it is an ideal protocol to govern the learning system effort known as the PPME.

Development of the PPME

The Pearce Projective Modeling Environment (PPME) was developed to facilitate the visualization of general linear modeling concepts on an IBM compatible computer. The backbone of the environment is a computer program that is used by students and instructors to demonstrate graphically a

[1]MathCAD version 2.5. MathSoft Inc., Cambridge MA, 1989 is a form free electronic spreadsheet that performs a wide variety of mathematical functions.

projection approach to the General Linear Model. Figure 1 contains a flowchart that outlines the basic flow of the program.

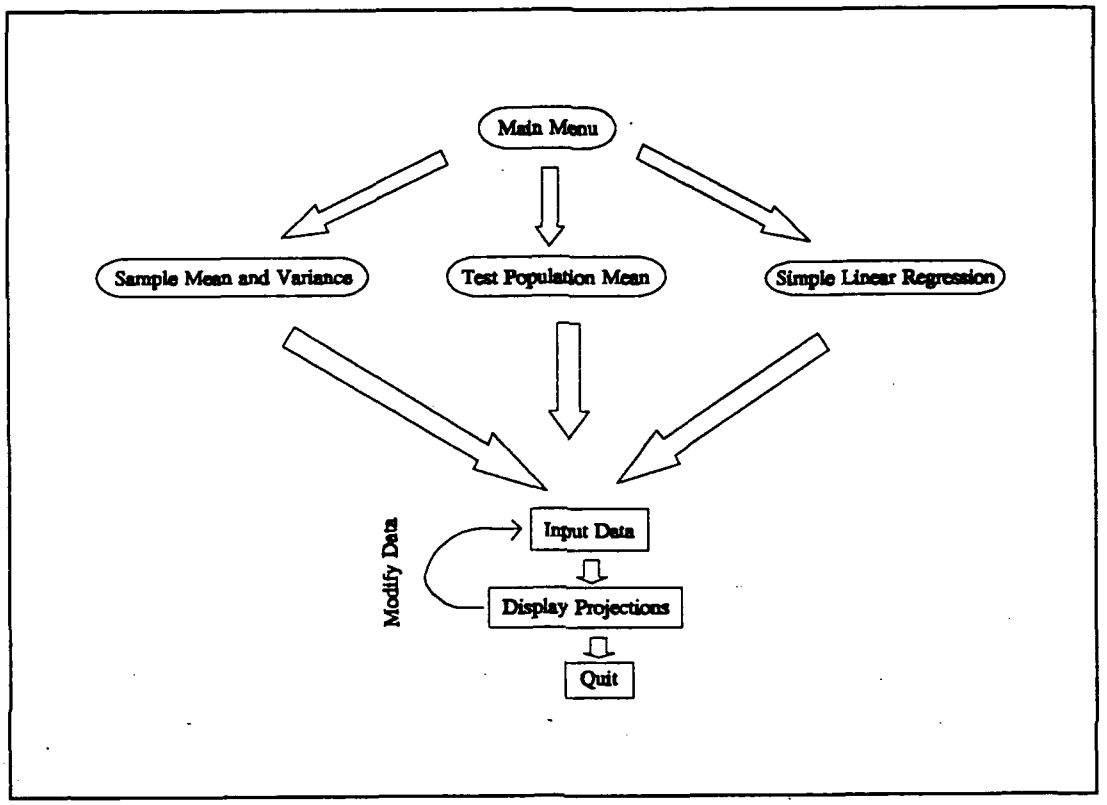

Figure 1. PPME General Flow Diagram

Once the program is run, the student can choose to experiment with the Sample Mean and Variance, the General Linear Test of the Population Mean, or Simple Linear Regression.

If the student chooses the Sample Mean and Variance option, he is asked for the response vector, $\underline{Y}$, and the design matrix, X. From this data the computer calculates the projection of $\underline{Y}$ onto X and the error vector

associated with that projection. Several other pieces of explanatory information are also provided and are discussed later in this chapter.

Alternatively, the student can opt to conduct a General Linear Test of the Population Mean. The selected module then requests specific information from the user relating to that chosen area. Next, inputs are processed and displayed. At this point the user can observe the data graphically by rotating the plotting axes to a preferred vantage point or by turning selected vectors on and off. By modifying the input data interactively, the student can see how changes to the input data affect the displayed geometric structures of the GLM. More details on the program follow later in this chapter and are accessible, during execution of the PPME, as on-line help screens.

Our next task will be to present the mathematical framework that serves as the formal foundation for a projective approach to the study of the GLM. Details concerning how the mathematical framework was transformed into a working interactive environment (known as the PPME) follow this discussion. The chapter concludes with an overview of the three specific scenarios that served as a test bed for evaluating the ability of the PPME to generate a meaningful learning environment. Finally, the four criterion which were used to make the evaluation are defined.

Chapter 4 presents six data sets, defined later in this chapter, to confirm the research hypotheses specified in Chapter 1. The PPME's ability to handle each data set is evaluated in terms of its *constructiveness, meaningfulness, livingness, and relatedness.* The constructiveness criterion will receive special attention in Chapter 4 since it is the criterion that is used to evaluate how well the PPME system facilitates student construction of new knowledge (at least from the student's perspective) about the GLM. The meaningfulness criterion is employed to assess the assimilatability of GLM concepts when the PPME is exercised. Assessing the PPME's capability to stimulate interest in the subject matter is the task assigned to the criterion of livingness. The PPME's ability to relate concepts of the GLM to some real world situation is evaluated by applying the criterion of relatedness.

Overall Mathematical Framework for the GLM. The concepts of the theory of the GLM can be developed from a geometric standpoint. In order to visualize this, consider the response vector, $\underline{Y}$, and the Estimation Space, $\underline{X}$, shown as:

$$\underline{Y} = \begin{bmatrix} Y_1 \\ \vdots \\ Y_n \end{bmatrix} \qquad \underline{X} = \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} \qquad n = 2 \text{ or } 3$$

The response vector $\underline{Y}$ resides in the sample space of the GLM. This research deals only with sample spaces of two or three dimensions. However, once the student understands the basic concepts involved in exercising the GLM, concepts displayed in two or three dimensions can be extended quite naturally to sample spaces with any number of dimensions. The design matrix, $\underline{X}$, is a column of ones with the same length as $\underline{Y}$.

The first concept to be demonstrated will be the estimation of the mean of $\underline{Y}$. The basic equation for least squares estimation of the mean is

$$\underline{Y} = \mu\underline{X} + \underline{\varepsilon} \tag{1}$$

The estimate of the mean of $\underline{Y}$, known as $\underline{\hat{Y}}$, is actually a projection of $\underline{Y}$ onto the Estimation Space, $\underline{X}$, which is represented by a column matrix of ones with the same number of rows as $\underline{Y}$. Since $\underline{\hat{Y}}$ is a projection onto $\underline{X}$, it must lie on $\underline{X}$; therefore, the Estimation Space must have a dimension of one. The remaining two dimensions contain the error vector that resides in what as known as the Error Space. The equations below show how to calculate the projection matrix, M; the estimate of the mean of $\underline{Y}$, $\underline{\hat{Y}}$; and the error vector, $\underline{e}$.

$$M = X[X^T X]^{-1} X^T$$

$$\underline{\hat{Y}} = \underline{M}\,\underline{Y}$$

$$\underline{e} = \underline{Y} - \underline{\hat{Y}}$$

The dimensions of the space are obvious to the student who views them graphically. The dimension of the sample space is equal to the sample size. Hence, the $\underline{\hat{Y}}$ vector will be plotted in two or three dimensions. The dimensions of the Estimation Space for the Least Squares of the Sample Mean is one since the Design Matrix, X has only one column. The error space must consist of the remaining dimensions (n-1=2) because the estimation and error space dimensions always sum to equal the dimension of the sample space (1 +(n-1)= n) .

The second concept to be examined is the estimation of the parameter vector $\underline{\beta}$ which is known as the $\underline{\hat{\beta}}$ vector, calculated as follows.

$$\underline{\hat{\beta}} = [\underline{X}^T \cdot \underline{X}]^{-1} \underline{X}^T \cdot \underline{Y} = \hat{\mu} \quad (a\ scaler)$$

The $\underline{\hat{\beta}}$ vector, in this case, will be the scaler estimate of μ, or $\hat{\mu}$

General Linear Test about the Population Mean of a Normally Distributed Random Variable. In the case of the General Linear Test for the Population Mean, we have as the Full Model,

$$Y_i = \beta_0 + \beta_1 X_i + \varepsilon_i \qquad i = 1, \cdots, n \tag{2}$$

and the Reduced Model,

$$Y_i = \beta_0 + \varepsilon_i \qquad i = 1, \cdots, n \tag{3}$$

Since this is a test, a null hypothesis and alternate hypothesis must be stated. These are, in general,

$$H_0: \mu = \mu_0 \quad versus \quad H_a: \mu \neq \mu_0$$

To actually conduct this test a value for μ_0 and a value for Type I error, α, must be specified.

Once these structures and entities to the PPME have been defined, the projection matrix M can be calculated. The projection matrix is then used to calculate $\underline{\hat{Y}}$. In order to complete the test, the next step is to calculate the estimated error (residual vector) $\underline{e}_F$ for the Full Model,

$$\underline{e}_F = \underline{Y} - \underline{\hat{Y}}$$

and the Reduced Model,

$$\underline{e}_R = \underline{Y} - E(\underline{Y})$$

Hence, E(Y) is simply the μ_0 multiplied by the Design Matrix, X.

$$E(\underline{Y}) = \mu_0 \cdot \underline{1}$$

The next step is to compare the squared lengths of the two estimated error vectors. These quantities are known as Error Sum of Squares of the Full Model, SSE_F, and Error Sum of Squares of the Reduced Model, SSE_R.

To determine if the null hypothesis should be rejected or not, a critical value, F_{CRIT}, is computed based on the F distribution and specified α. This is compared to F^*, where

$$F^* = \frac{SSE_R - SSE_F}{df_R - df_F} \div \frac{SSE_F}{df_F}$$

and df_F and df_R are the degrees of freedom for the Full and Reduced Models, respectively. If F^* is greater than F_{CRIT}, the null hypothesis is rejected and the alternate hypothesis is accepted; otherwise, the null hypothesis cannot be rejected. The estimates of $\underline{\beta}$ for both Full and Reduced Models are calculated as shown:

$$\underline{\hat{\beta}} = [\underline{X}^T \cdot \underline{X}]^{-1} \underline{X}^T \cdot \underline{Y}$$

Linear Simple Regression to Estimate E(Y /X). In this application of the GLM, the PPME tests whether the slope parameter $\beta_1=\beta_{10}$, assuming $\underline{\hat{\beta}}_1$ is normally distributed.

Since this is also a test, a level of Type I error, α, must be given. As with the previous two cases the design matrix, X, and response vector, $\underline{Y}$, must then be input. In this case the design matrix, X, has two columns. While the sample space still contains three dimensions, the Estimation Space, which is based on the number of columns in X, has dimension two. As a result, the error space must lie in a subspace of dimension one.

$$\underline{Y} = \begin{bmatrix} Y_1 \\ \vdots \\ Y_n \end{bmatrix} \qquad X = \begin{bmatrix} 1 & X_1 \\ \vdots & \vdots \\ 1 & X_n \end{bmatrix} \qquad n = 3$$

In this test, we also have the Full Model,

$$Y_i = \beta_0 + \beta_1 X_i + \varepsilon_i \qquad i = 1, \cdots, n \tag{4}$$

and the Reduced Model,

$$Y_i = \beta_0 + \varepsilon_i \qquad i = 1, \cdots, n \tag{5}$$

We must next establish our null hypothesis, H_0, and alternate hypothesis, H_a:

$$H_0: \beta_1 = \beta_{10} \quad versus \quad H_a: \beta_1 \neq \beta_{10}$$

To do this, a value for Type I error, α, needs to be selected.

Once these structures and entities of the PPME have been defined, the projection matrix M can be calculated. The projection matrix is then used to calculate $\underline{\hat{Y}}$. In order to complete the test, the next step is to calculate the estimated error (residual vector) $\underline{e}_F$ for the Full Model,

$$\underline{e}_F = \underline{Y} - \underline{\hat{Y}}$$

and the Reduced Model,

$$\underline{e}_R = \underline{Y} - \underline{E}(Y)$$

Hence, E(Y) is simply the μ_0 multiplied by the Design Matrix, X.

$$E(\underline{Y}) = \mu_0 \cdot \underline{1}$$

The next step is to compare the squared lengths of the two estimated error vectors. These quantities are known as Error Sum of Squares of the Full Model, SSE_F, and Error Sum of Squares of the Reduced Model, SSE_R.

To determine if the null hypothesis should be rejected, a critical value, F_{CRIT}, must be computed based on the F distribution and specified α. This is compared to F^*, where

$$F^* = \frac{SSE_R - SSE_F}{df_R - df_F} \div \frac{SSE_F}{df_F}$$

and df_F and df_R are the degrees of freedom for the Full and Reduced Models, respectively. If F^* is greater than F_{CRIT}, the null hypothesis is rejected and the alternate hypothesis is accepted; otherwise, the null hypothesis cannot be rejected. The estimates of $\underline{\beta}$ for both Full and Reduced Models are calculated as shown:

$$\underline{\beta} = [\underline{X}^T \cdot \underline{X}]^{-1} \underline{X}^T \cdot \underline{Y}$$

Computer Program Development. The mathematics involved in the geometric approach to the GLM have been around for many years. The revolutionary part of this research is the creation of the PPME which uses an interactive three-dimensional graphics display to portray the vector spaces associated with the GLM and computer. With the PPME the user can visually study the vector spaces and subspaces associated with the GLM from any viewpoint.

In order to facilitate the design of the program, the computer program was broken down into three logical modules with one for the Sample Mean and Variance, another for the General Linear Test of the Population Mean, and a third for the Simple Linear Regression. Borland Company's Turbo Pascal v6.0 was chosen as the programming language because of the modular design capabilities of the Pascal language. A commercial set of subroutines, AcroMolé, by AcroSpin, Inc was used to incorporate a capability for interactive graphics into the program.

In order to keep the programming to a minimum, most of the code was written so that each of the three modules could reuse the same code. Because of the extensive reuse of code, most of the commands within each module are very similar. For example, in each of the modules "F1" accesses help and Alt-Y modifies the Y matrix. It is believed that by reducing the

time required to learn each module, the student is given a better opportunity to work with the mathematics.

In support of this philosophy the output of each of the modules was designed to be as similar as possible. Specifically, the coding for the response vector and design matrix reside at the same location in each module. More importantly, the color of the $\underline{Y}$ vector and other vectors does not change from scenario to scenario. By allowing the student to concentrate on the mathematics, his understanding of the concepts of the GLM is enhanced.

While the PPME is simple to use, it is not designed as a tutorial. Therefore, no conceptual information is provided through its auspices. The student needs to be guided by (1) the governance provided by the VVAM, (2) the learning heuristics developed during this research and by (3) a GLM competent instructor. The PPME was designed to be used this way because it is believed that the required conceptual information is best assimilated through a VVAM driven interaction of student and teacher.

Once executed, the program allows the user to choose any one of the three major topics listed in Figure 2. If execution of the Sample Mean and Variance or the General Linear Test of the Population Mean routine is requested, the user is then asked to enter the size of the sample: n=2 or n=3. The third module, which assists in Simple Linear Regression, assumes the

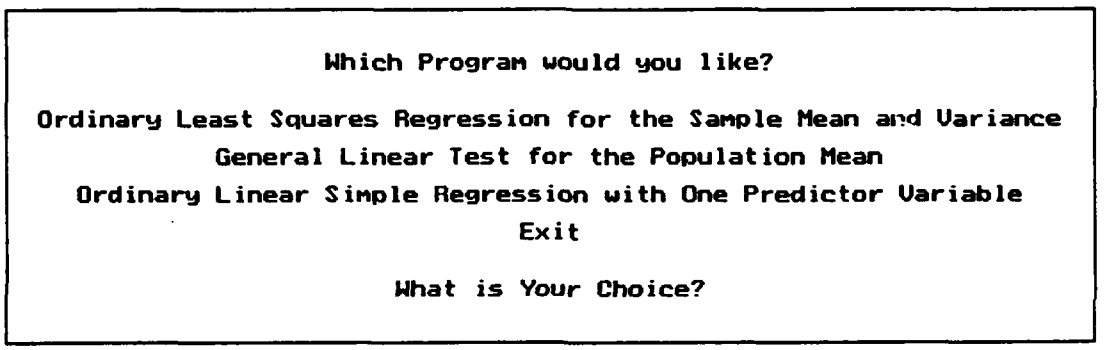

Figure 2. Main Menu Screen

sample size is n=3.

Sample Mean and Variance. If the user chooses the Sample Mean and Variance option, he is asked to input the observed values for the response vector, $\underline{Y}$, and the design matrix, X. Since the design matrix should be a column of ones, it is set to ones by default. The input format for the matrices is based on MathCAD's matrix input format because the students who will use the PPME at the Air Force Institute of Technology (AFIT) are familiar with MathCAD. Numbers are limited to a size of four digits (including the decimal point) because, for educational purposes, this level of precision should be sufficient. Once the matrix inputs are entered, the user must press the 'F10' key to indicate that he is finished.

After the user enters these initial inputs, the program calculates several different entities. The projection matrix is calculated and displayed along with the $\underline{\beta}$ vector and $\underline{\hat{Y}}$ vector. The computer then plots the response

vector, $\underline{Y}$, the Design Matrix vector, $\underline{X}$, the projection of $\hat{\underline{Y}}$ onto the Estimation Space, and the Error vector, $\underline{e}$, in the lower left hand portion of the screen. The dimensions of the Sample, Estimation, and Error Space are also displayed. So that the student can visualize the explanatory power of his selected response vector and design matrix, the Regression Sum of Squares (SSR) and Error Sum of Squares (SSE) are also displayed as a percentage of the Total Sum of Squares (SSTO) which is the total unexplained variation in $\underline{Y}$. These are plotted as an area to facilitate the visualization of the explanatory power of the response vector when the student compares the SSR to the SSE. In addition to the plot, the actual values of SSR, SSE, and SSTO are also provided. Figure 3 gives the reader an idea of how the data is displayed. Unfortunately, the static figures printed in this thesis cannot portray the extraordinary dynamics the actual program is able to facilitate. Additionally, on the actual computer display, each of the vectors are easily distinguished by color and by selective deletion.

General Linear Test for the Population Mean. If the user chooses the second option, then he will see the screen displayed in Figure 4. He is asked to enter both the response vector, $\underline{Y}$, and the Design Matrix, $\underline{X}$. Since this is a test, the user is then prompted for a μ_0 and a level of Type I

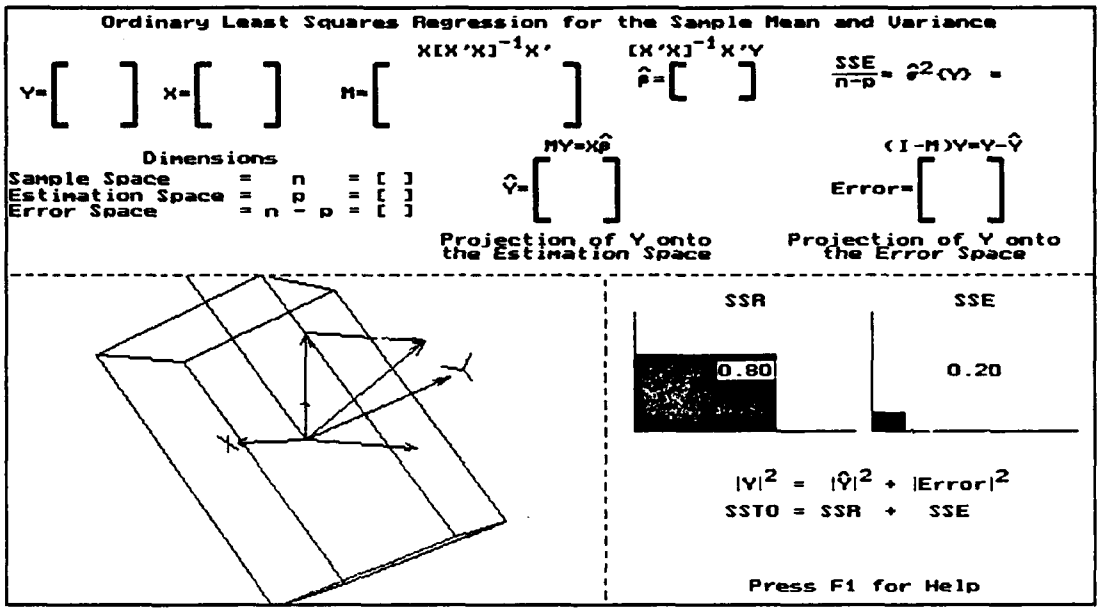

Figure 3. Sample Mean and Variance Screen

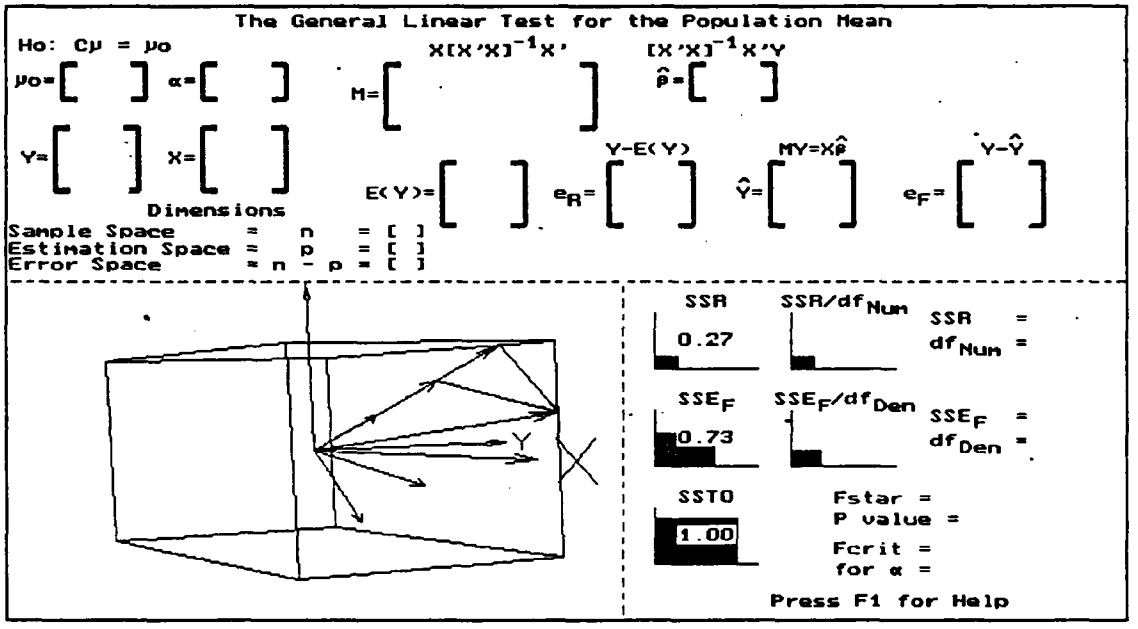

Figure 4. General Linear Test for Population Mean Screen

error, α. A default of 0.05 is provided for α and a column of ones is the default for the design matrix.

After the user completes these inputs the program begins to calculate the projected vectors. The projection matrix is first calculated and displayed, as are $\underline{\hat{\beta}}$ and $\underline{\hat{Y}}$. The computer then plots the response vector, $\underline{Y}$, the Design Matrix, $\underline{X}$, the projection of $\underline{Y}$ onto the Estimation Space, $\underline{\hat{Y}}$, and both the Reduced ($\underline{e}_R$) and Full ($\underline{e}_F$) Error vectors, in the lower left hand portion of the screen. The dimensions of the Sample, Estimation, and Error Space are also displayed. So that both the error and the explanatory power of the Full Model can be seen, the Regression Sum of Squares (SSR) and Error Sum of Squares (SSE_F) are displayed as a percentage of the Total Sum of Squares (SSTO) which is the total unexplained variation in $\underline{Y}$. The SSR and SSE_F are then divided by their respective degrees of freedom and then plotted again. SSE_F and SSR are computed as shown below:

$$SSE_F = Y - \hat{Y}$$

$$SSR = |\hat{Y}|^2$$

P values are also generated to help the student decide whether to accept the alternate hypothesis H_a, or reject the null hypothesis H_0.

The dimensions of the sample space, and each subspace, will become more obvious as the student views them graphically. The dimension of the sample space is based on sample size and is either two or three dimensions. Relevant vectors are plotted in two or three dimensions.

Simple Linear Regression. When the student chooses the third option, Simple Linear Regression with One Predictor Variable, he will be asked to enter the response vector, $\underline{Y}$, and design matrix, X. Because the sample size was defined to be 3, the design matrix now has two columns. The first column is all ones. Since this is a test, the user is then prompted for the value of $\underline{\beta}_{10}$ and level of Type I error, α. A default of 0.05 is provided for α. Once the data is input several calculations must be made before it can be displayed as shown in Figure 5.

The program first calculates the projection matrix and then displays it along with $\underline{\hat{\beta}}$ and $\underline{\hat{Y}}$. The computer then plots the response vector, $\underline{Y}$, the Design Matrix vector, $\underline{X}$, the projection of $\underline{Y}$ onto the Estimation Space, $\underline{\hat{Y}}$ and both the Reduced ($\underline{e}_R$) and Full ($\underline{e}_F$) Error vectors, in the lower left hand portion of the screen. The dimensions of the Sample, Estimation, and Error Space are also displayed. As was done in the previous test, both the error and the explanatory power of the Full Model can be seen as the SSR and SSE_F are displayed as a percentage of the SSTO. The SSR and SSE_F are then divided by their respective degrees of freedom and plotted again. SSE_F and SSR are computed as shown below:

$$SSE = Y - \hat{Y}$$

$$SSR = |\hat{Y}|^2$$

P values are also generated to help the student decide whether to accept the alternate hypothesis H_a, or reject the null hypothesis H_0.

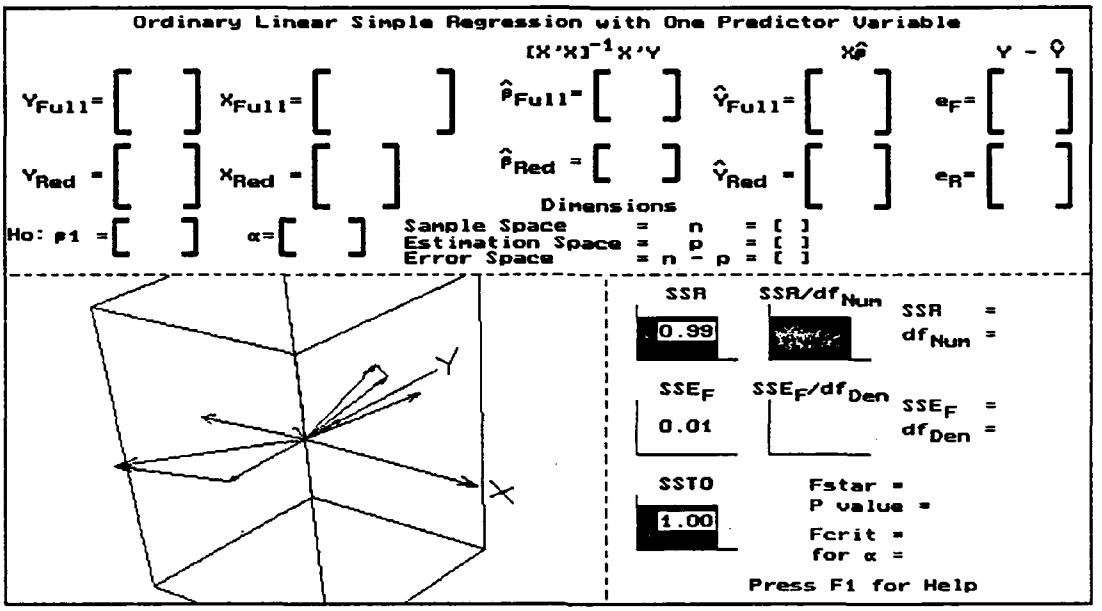

Figure 5. Simple Linear Regression Screen

Specific Scenarios

The purpose of the PPME learning environment is to encourage the student to explore the various aspects of the General Linear Model. In support of this goal, six data sets were selected to help demonstrate the ability of the PPME, employed under the guidance of the VVAM, to facilitate a meaningful learning environment for studying the GLM.

The six data sets have different ranges of variability. One relative measure of variability, the Coefficient of Variation, represents the ratio of the standard deviation to the mean, and is calculated as follows:

$$\textit{Coefficient of Variation} = \frac{\sigma}{\mu}$$

By using such diverse data sets it was possible to demonstrate how the PPME can be employed to facilitate meaningful interaction between the student and instructor, who as co-creators participating in a meaningful learning process are required to construct concept maps of their personal knowledge about the General Linear Model. (7:130-132)

The six data vectors and their specific elements and coefficients of variation are listed in Table 1.

Table 1. Data Sets

Set No	General Case	Specific Case	Mean	Standard Deviation	Coeff of Var
0	$[0,0,0]^T$	$[0,0,0]^T$	0	0	⌀
1	$[-c,0,c]^T$	$[-2,0,2]^T$	0	2	∞
2	$[c,c,c]^T$	$[2,2,2]^T$	2	0	0
3	$[c,c+\alpha,c+2\alpha]^T$	$[2,4,6]^T$	4	2	50%
4	$[c,c+\alpha,c+11\alpha]^T$	$[2,4,24]^T$	10	12.2	122%
5	$[-3c,c,2c]^T$	$[-6,2,4]^T$	0	5.29	∞

By allowing the student to observe the effect these different data sets have on the visible geometry of a particular linear model, a better

understanding of the relationships of concepts and entities forming the structures of the GLM can be obtained.

Three scenarios were proposed to provide a context in which students and teachers could use the PPME to explore the concepts of the GLM under governance by the VVAM protocol.

Scenario One. The first scenario suggests the General Linear Model be used to carry out an Ordinary Least Squares Estimation of the Sample Mean and Variance. Data sets 1 through 5 were employed as the PPME's efficacy and ability to facilitate meaningful learning under this scenario was evaluated.

The model required in this case is a subset of the GLM in which the Estimation Space is a column of ones as shown below,

$$\underline{Y} = \mu \underline{1} + \underline{e}$$

The first assumption this model makes is that the error vector is independent and normally distributed with an expected value of 0. In this first scenario, the inputs are the response vector, $\underline{Y}$ and the Estimation Space, $\underline{X}$. From these inputs the following calculations are made: M, the projection matrix; $\underline{\beta}$ the estimate of the regression coefficients; $\underline{\hat{Y}}$, the estimate of the mean; $\hat{\sigma}^2_Y$, the estimate of the variance of $\underline{Y}$; $\underline{e}$, the estimated error vector;

SSTO, the total sum of squares; SSR, the regression sum of squares; and SSE, the error sum of squares.

After calculation, several vectors, $\underline{X}$, $\underline{Y}$, $\underline{\hat{Y}}$, and $\underline{e}$ are displayed graphically for the teacher and student to discuss. The Estimation Space, defined by $\underline{X}$, is one-dimensional in this scenario.

Scenario Two. To conduct a General Linear Test about the population mean of a normally distributed random variable both Full and Reduced Models must be specified. In evaluating the PPME's capacity to orchestrate meaningful learning under this second scenario, data set 3 was employed.

The Reduced Model associated with the null hypothesis, H_0, can be represented as shown:

$$Y = \mu_0 1 + \varepsilon$$

and the error vector for the Reduced Model is computed $\varepsilon = Y - \mu_0 1$. Since $E(Y) = \mu_0 1$, the estimated error vector is calculated as

$$e_R = Y - E(Y)$$

The alternate Hypothesis, H_a, is identified with the Full Model and is represented as shown:

$$Y = \mu 1 + \varepsilon$$

so that when μ is estimated by $\hat{\mu}$, the error vector for the Full Model is computed as shown:

$$e_F = Y - \hat{Y} = Y - \hat{\mu}\,1$$

In this scenario, the inputs are μ_0, the response vector, $\underline{Y}$, the level of Type I error, α, and the Estimation Space, $\underline{X}$, which is a column of ones. From these inputs, M, $\underline{\hat{\beta}}$, $\underline{\hat{\mu}}$, $\hat{\sigma}^2_Y$, and error reduced, $\underline{e}_R$; error full, $\underline{e}_F$; SSR, SSE, and SSTO can be computed.

From these calculations several vectors, $\underline{Y}$, $\underline{\hat{Y}}$, and $\underline{e}$, as well as $E(\underline{Y})$, $\underline{X}$, $\underline{e}_R$, and $\underline{e}_F$ are graphically displayed for the teacher and student to discuss. Through the geometry of the GLM we know that the length of the error vector for the Reduced Model squared is equal to the length of $\underline{\hat{Y}}$ squared plus the length of the error vector for the Full Model squared, which is written as:

$$|\underline{e}_R|^2 = |\underline{\hat{Y}}|^2 + |\underline{e}_F|^2$$

hence the regression sum of squares is equal to the length of $\underline{\hat{Y}}$ squared or the length of the reduced error vector squared minus the full error vector squared as shown below:

$$SSR = |\hat{Y}|^2$$

$$SSR = |e_R|^2 - |e_F|^2$$

Scenario Three. When the General Linear Model is used to conduct a Simple Linear Regression to estimate $E(Y \mid X)$ or make a test about the slope parameter β_1, we assume that its estimator, $\hat{\beta}_1$, is normally distributed. As in the previous scenario, the concepts motivated by this General Linear Test require an understanding of hypothesis testing and the Full and Reduced Models. Data set 3 used along with the Design Matrix X_2. Employment of Design Matrix X_1 is left to a future researcher. These two design matrices were chosen to represent Estimation Spaces with an evenly distributed, and positively skewed, independent variable.

$$X_1 = \begin{bmatrix} 1 & 2 \\ 1 & 4 \\ 1 & 6 \end{bmatrix} \qquad X_2 = \begin{bmatrix} 1 & 2 \\ 1 & 3 \\ 1 & 10 \end{bmatrix}$$

The Reduced Model associated with the null hypothesis, H_0, is represented as shown:

$$Y = 1[\beta_o] + \varepsilon$$

where β_0 is unknown. The error for the Reduced Model is given as the difference between the Y_{Red} and the $\hat{Y}_{Red}$ and is written as shown:

$$e_R = Y_{Red} - \hat{Y}_{Red}$$

where $Y_{Red} = Y - X_{(i,2)}\beta_{10}$ and $Y_{Red} = X\hat{\beta} = \underline{1}\hat{\beta}_0$.

The Full Model is associated with the alternate hypothesis, H_a, and is represented as shown:

$$Y = X\underline{\beta} + \varepsilon$$

If we estimate ß with $\underline{\hat{\beta}}$, the error for the Full Model is given as the difference between the Y_{Full} and the $\hat{Y}_{Full}$ and is written as shown:

$$e_R = Y_{Full} - \hat{Y}_{Full}$$

Evaluation Criteria

Once details for each of the scenarios were determined, four criterion for evaluating the efficacy of the PPME's ability to orchestrate meaningful learning experiences with the concepts of the GLM within the context provided by the three scenarios were formulated. These four criteria were invoked to evaluate this efficacy and were labeled: Constructiveness, Meaningfulness, Livingness, and Relatedness. It is this researcher's belief that all four criteria must be considered to adequately assess the pedagogical value of any educating event. The analysis in Chapter 4 will give primary emphasis to the constructiveness criterion because, in conjunction with the other criteria, it is both necessary and sufficient to the attainment of a

meaningful learning experience. Each criterion was measured on a scale that can be treated as ordinal in any future formal evaluation exercise.

The *Constructiveness* criterion evaluates the PPME's ability to orchestrate the construction of new knowledge as the student and teacher interact with the geometric display and the student attempts to assimilate key concepts of the GLM with the teacher's assistance. It also is used to measure the extent to which the student is able to manage his own learning. Its scale represents a continuum of constructiveness with designated extreme values of *dormant*, indicating no constructive ability, through *constructive*, indicating all new knowledge is generated by the student.

The scale of *Meaningfulness* attempts to assign a measure that documents the PPME's capacity to encourage students to link and subsume new concepts to concepts previously assimilated during prior learning sessions. It's scale measures a continuum of meaningfulness with designated extreme values of *rote*, implying mindless memorization of concepts, through *meaningful*, suggesting complete assimilation and subsumption of all new concepts introduced during the evaluated session.

Livingness, the third criterion, is a proposed measurement of the ability of the PPME, during any encounter with the GLM, to make concepts come alive for the student through dynamic visualizations of the subject matter. This scale measures a continuum of livingness with designated

extreme values of *non-living*, indicating a moribund encounter with the subject matter, through *living*, indicating an evolving and deepening relationship with the concepts of the GLM.

Finally, the criterion of *Relatedness* is used to evaluate the PPME's ability to facilitate a concrete awareness of the applicability of the GLM to real world problems. Students tend to be more inspired when they can see links to the field while studying theory in the classroom. This measure is used to record the PPME's ability to facilitate, under the VVAM governance and the teacher's watchful eye, such linkage in the mind of the student.

These four measures clearly impact one another and represent a system of criteria. Figure 6 considers their mutual relationships and their tie to the real world via three of the four commonplaces of any educating event (Teacher, Student, Governance, and Curriculum) and the real world. It should be noted that the individual continuums of each criterion relate specifically to one of several possible two-way interactions diagrammed by Figure 6.

The onus is on the teacher to make the curriculum interesting for the student. However, responsibility for learning is entirely on the shoulders of the student and may or may not prove constructive depending on the relationship between the student and the curriculum. A meaningful educating event, while not exclusively the result of the student and teacher

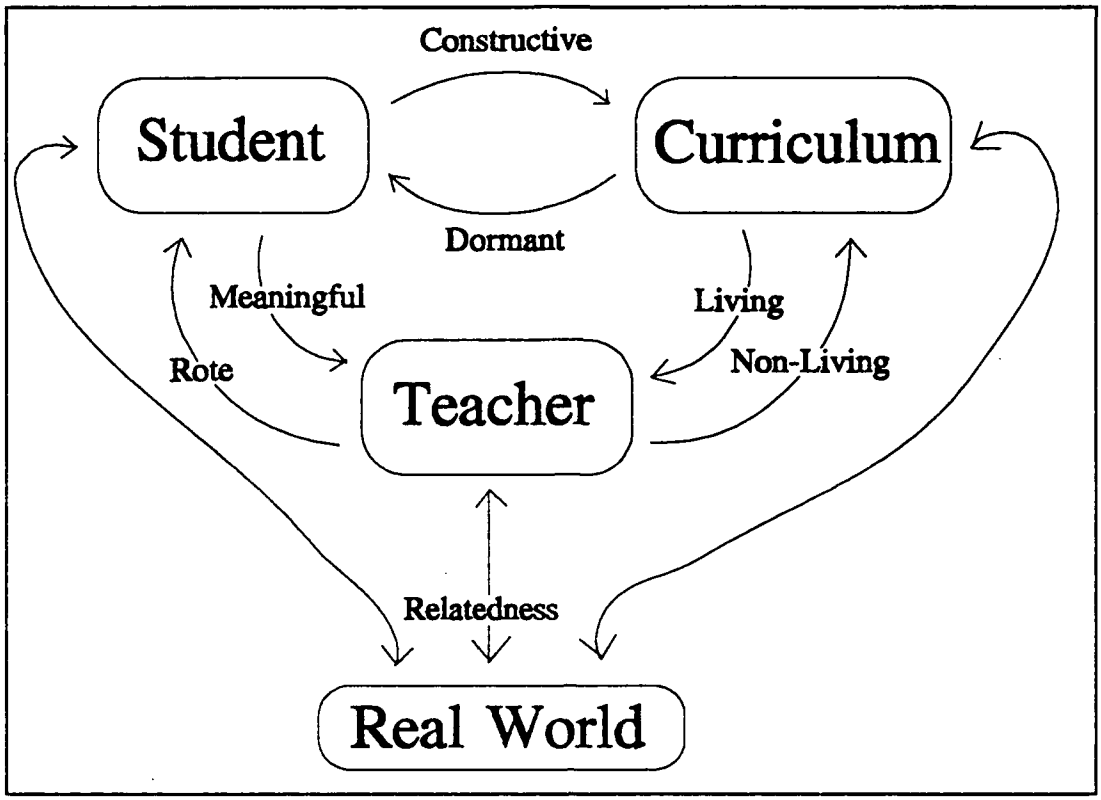

Figure 6. Relationship Between Evaluation Criteria and Commonplaces

interaction, typically occurs only when the student and teacher serve each other as co-operating coequals during the process of constructing new knowledge that always characterizes any meaningful learning activity. Perhaps the most fundamental postulate suggested by Figure 6 is that while the rich dynamics between the three commonplaces are necessary for meaningful learning to take place, each of the three commonplaces, under the governance of the VVAM protocol, must possess a credible and continuous relationship to the real world if they are to serve as a sufficient basis for the manifestation of a constructive and meaningful educating event.

In Chapter 4 the VVAM, the PPME and the six data sets presented in this chapter are employed within the context of the three scenarios previously described to allow dynamic visualization of GLM concepts associated with each scenario. The four criterion are used to evaluate the PPME's efficacy for creating a meaningful learning environment in anticipation of future research efforts that would conduct a complete and formal evaluation of the PPME's use within a much broader domain of topics associated with the GLM and across a larger student population taking course work in the GLM.

IV. Analysis and Results

To assess the efficacy of the Pearce Projective Modeling Environment's (PPME) ability to orchestrate meaningful and self-managed learning activities, each scenario introduced in Chapter 3 was evaluated by exercising the PPME under VVAM governance using one or more of the data sets (response vectors) which were presented in Chapter 3.

The evaluation consisted of executing the PPME with each scenario using specific data sets determined from the generalized data sets presented in Table 1 of Chapter 3. The specific data sets were obtained using $c=2$, and $\alpha=2$.

The first scenario uses data sets one through five since a response vector of all zeros merely suggests the possibility of sampling a system's response to settings of a particular variable. This is continued for each of three scenarios. The last two scenarios test null hypotheses H_0: $\mu_0=2$ and H_0:$\beta_1=0$, using data sets one and three. This resulted in five analyses for scenario one, two analyses for scenario two and one analysis for scenario three.

Once the PPME was executed for a particular scenario, the analysis involved graphically displaying the GLM structures produced by the PPME and making observations to compare and contrast differences and similarities

of the GLM's output generated in response to the entering of various data sets.

Results of the analyses were evaluated with respect to the four evaluation criteria introduced in Chapter 3, Constructiveness, Meaningfulness, Livingness, and Relatedness, with special emphasis being placed on the level of constructiveness attained under each scenario.

Sample Mean and Variance

Scenario 1 involved using the General Linear Model and Ordinary Least Squares to estimate the Sample Mean and Sample Variance. As stated previously, the model for this scenario is:

$$\underline{Y} = \mu \cdot \underline{1} + \underline{\varepsilon}$$

In this equation, both $\underline{\mu}$ and $\underline{\varepsilon}$ are theoretical values and therefore are not displayed. They are estimated by the $\underline{\hat{Y}}$ vector and the Error vector in the PPME. The $\underline{1}$'s vector is entered as the Design Matrix $\underline{X}$ and the $\underline{Y}$ vector as produced representing a particular data set.

The output of the PPME is displayed in Figure 7. It consists of the response vector $\underline{Y}$ and Design Matrix $\underline{X}$ which are both input by the user. The projection matrix M, which is also displayed, is used to calculate $\underline{\hat{Y}}$, the projection of $\underline{Y}$ onto the Estimation Space, $\underline{X}$, and the Error vector, the

projection of $\underline{Y}$ onto the Error Space. The PPME displays the entered and computed values in an interactive three-dimensional plot. The SSR and SSE are computed and displayed as a percentage of SSTO above their non-normalized values. The calculated estimates of ß and variance are also displayed.

The first data set evaluated has the general form of [-c,0,c] and the specific form of [-2,0,2] and is shown in Figure 7.

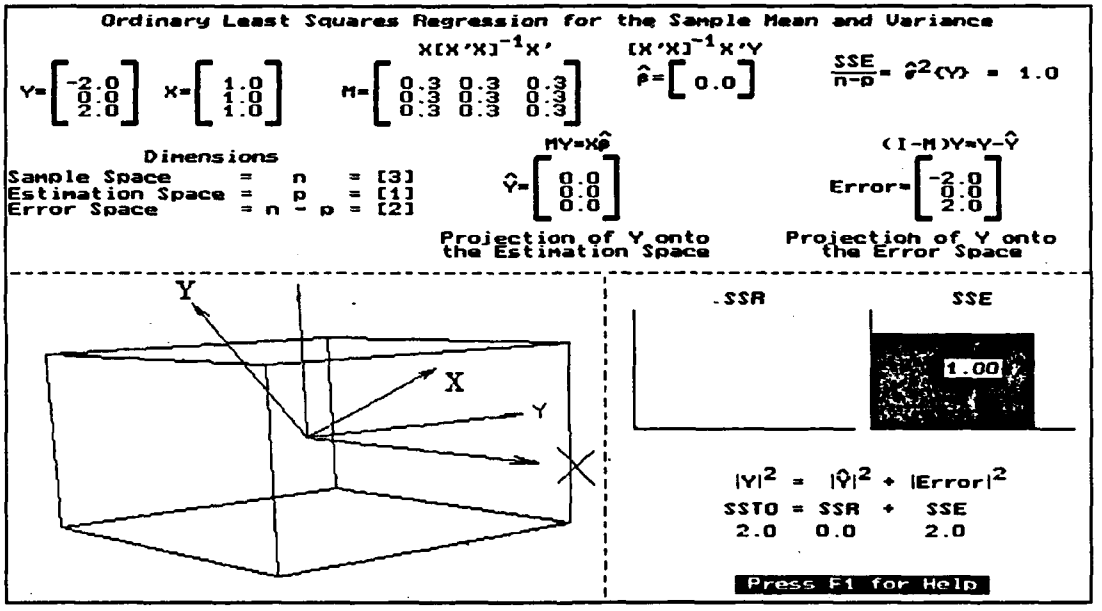

Figure 7. Data Set 1

The first thing to observe is the relationship between the projection of $\underline{Y}$ onto the Estimation Space, $\underline{\hat{Y}}$, and the estimated error vector, $\underline{e}$. When the angle between the vector $\underline{\hat{Y}}$ and $\underline{Y}$ is small, it is an indication that the estimator $\underline{\hat{Y}}$ has high explanatory value. However, data sets 1 [-2,0,2] and 5

[-6,2,4], shown in Figure 7 and Figure 8, both have a mean of 0 and a coefficient of variation of infinity. Since the zero vector is orthogonal to every other vector in a vector space and since $\hat{\underline{Y}} = \underline{0}$, the correlation of $\hat{\underline{Y}}$ to $\underline{Y}$ is zero. This lack of correlation indicates a lack of explanatory power,

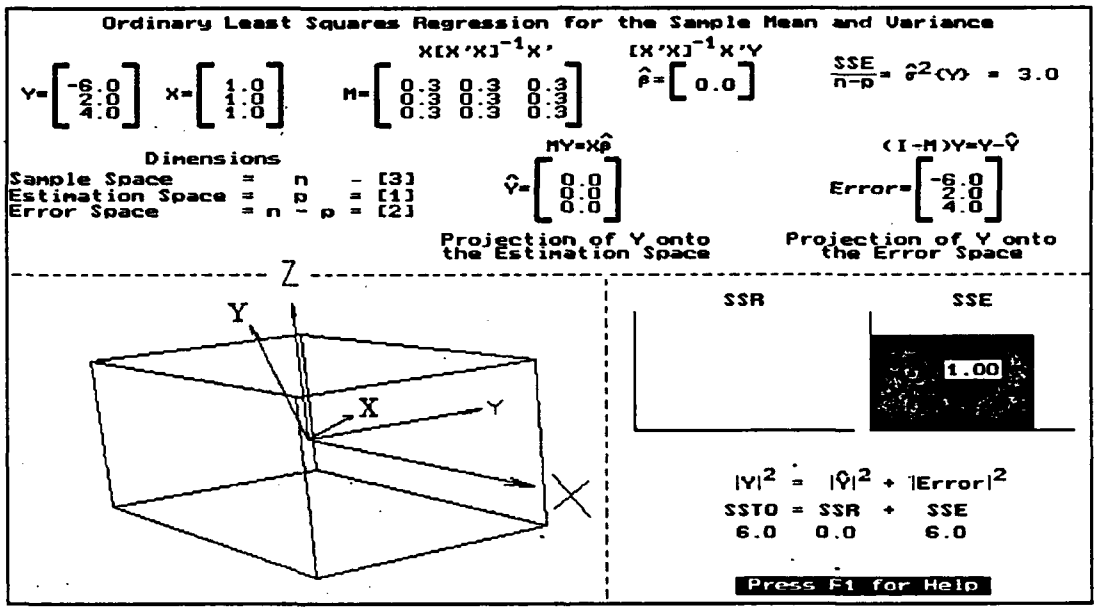

Figure 8. Data Set 5

and as expected, the PPME shows an SSR of 0 and the SSE of 1. While the estimate is perfectly correct, virtually none of the variation in the original data is explained by the estimating process. The error vector $\underline{e}$ is the difference between $\underline{Y}$ and $\hat{\underline{Y}}$ and since $\hat{\underline{Y}} = \underline{0}$, the error vector was equal to $\underline{Y}$. As shown in Figure 7 and Figure 8, the Error Sum of Squares, SSE, is 1

which means it equals the Total Sum of Squares. Thus the data from these two sets yields only error and is of no explanatory value.

Data set 2 [2,2,2], shown in Figure 9, was similar to 1 and 5 except

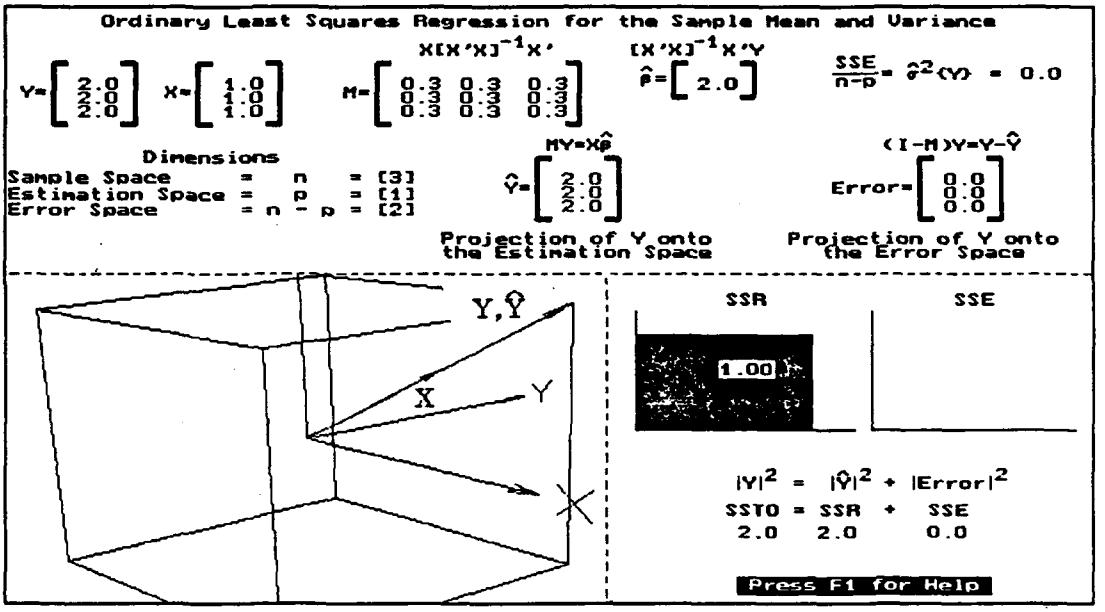

Figure 9. Data Set 2

the error vector, $\underline{e}$, is equal to zero and $\underline{Y} = \underline{\hat{Y}}$. Because of this perfect collinearity, the value of r^2 (the Coefficient of Determination) is 1 which means SSR equals the Total Sum of Squares.

A student can verify that $\underline{Y}$ and $\underline{\hat{Y}}$ are equal by looking at the values in their vectors in Figure 9 but this can also be verified graphically through the PPME plot of the vector space. When the student presses the "F2" key, a box like Figure 10 appears and indicates to the student what key to press to turn individual vectors on and off. By pressing Alt-Y, the $\underline{Y}$ vector

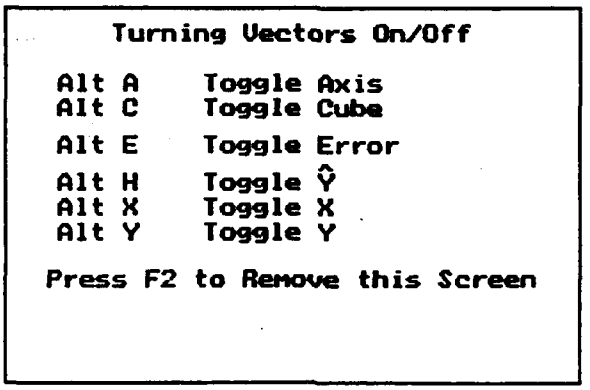

Figure 10. Vector Toggle Help

disappears and the $\underline{\hat{Y}}$ vector is visible. This capability is used during the analyses of scenario 2 and scenario 3.

Data sets 3 [2,4,6] and 4 [2,4,24] provide more conventional observations because neither their mean nor variance equals zero as indicated by their coefficients of variation of 50% and 122%. The vectors generated by these data sets are displayed in Figure 11 and Figure 12. Since most investigations in which regression is used to study variability, these two sets have practical value. Independence is visually demonstrated in Figure 11 and Figure 12. The orthogonality between the error vector, $\underline{e}$, is apparent since they are at right angles. The figures also show that the magnitude of the error vector of data set 3 is greater than that of data set 4. This is verified by noting the difference in their proportions of unexplained error (0.44 vs 0.33). Since data set 3 has a smaller Coefficient of Variation than set 4, it might be expected to have more explanatory power.

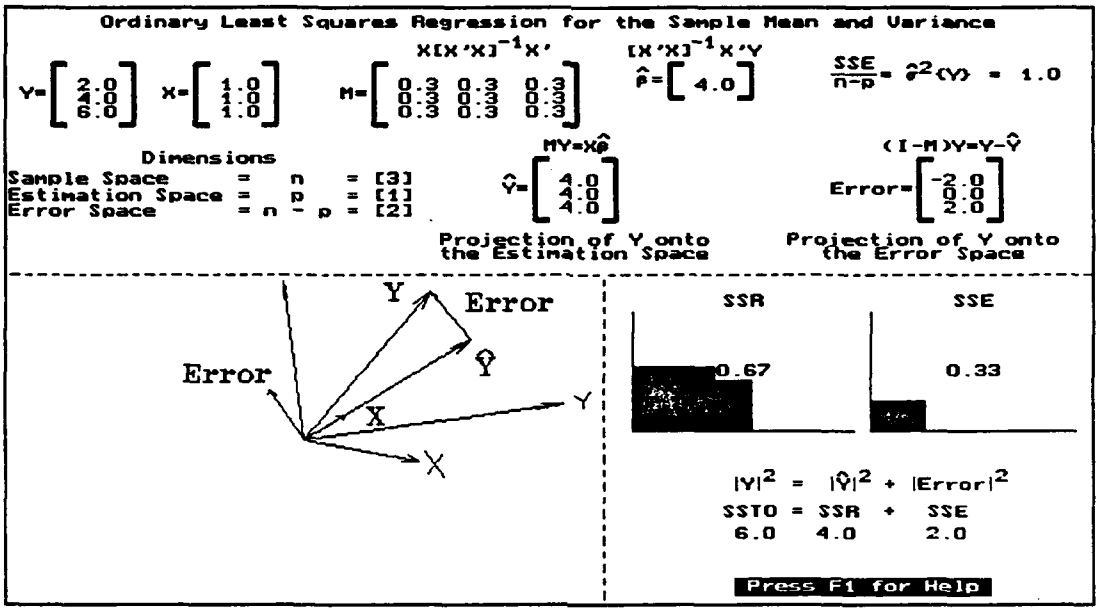

Figure 11. Data Set 3

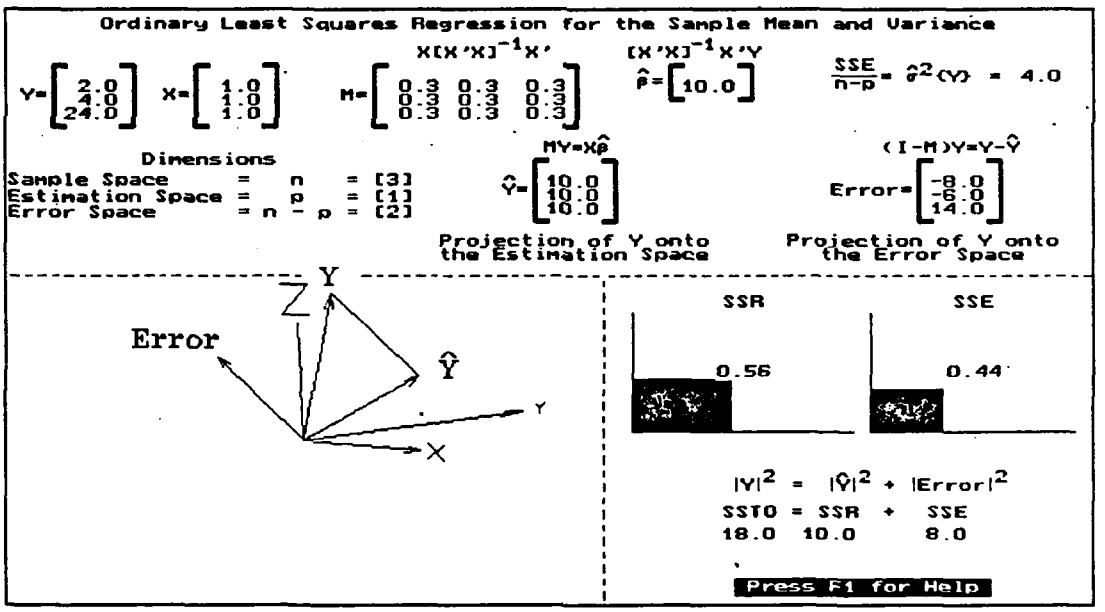

Figure 12. Data Set 4

The ability to enter and modify any data set and to observe the angle between the $\underline{Y}$ vector and the $\underline{\hat{Y}}$ vector and its relationship to the magnitude

of SSR and SSE to the angles magnitude is a solid confirmation of the constructive value of the pedagogy facilitated by the PPME.

General Linear Test about the Population Mean of a Normally Distributed Random Variable

The object of this scenario is to compare the error vectors from the Reduced Model which is associated with the null hypothesis, H_0, and represented as shown:

$$Y = \mu_o 1 + \varepsilon$$

and the Full Model which is associated with the alternate hypothesis, H_a, and represented as shown:

$$Y = \mu 1 + \varepsilon$$

In this scenario, tests will be conducted on two data sets. Data set 1 [-2,0,2] has a mean of zero and a standard deviation of two which drives the Coefficient of Variation to infinity. Data set 2 [2,4,6] is the second response vector and has a mean of four, a standard deviation of two and a Coefficient of Variation of 50%.

The first test consisted of establishing the null hypothesis, H_0: $C\mu = \mu_0$, in which μ_0 equals 2. The images provided by the PPME for this data set are shown in Figure 13. The projection of $\underline{Y}$ onto $\underline{X}$ for set 1, $\underline{\hat{Y}}$

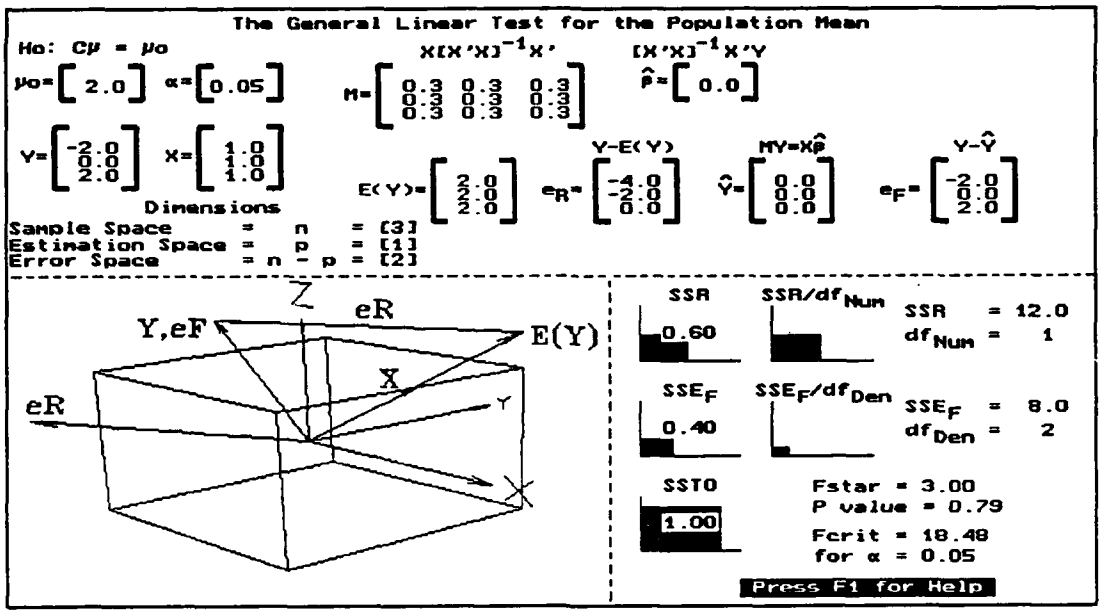

Figure 13. Data Set 1, $\mu_0=2$

is the zero vector. The Full Model error vector ($\underline{e}_F$) is the difference between Y and $\underline{\hat{Y}}$, therefore $\underline{e}_F = \underline{Y}$. The Reduced Model error vector ($\underline{e}_R$) is the difference between Y and E($\underline{Y}$). The significance of these two error vectors is that their lengths form the basis for the F statistic below:

$$F^* = \frac{|\underline{e}_R|^2 - |\underline{e}_F|^2}{df_R - df_F} \div \frac{|\underline{e}_F|^2}{df_F}$$

As stated earlier, the PPME allows the user to toggle vectors on and off. By turning off all of the vectors except $\underline{e}_F$ and $\underline{e}_R$, Figure 14 illustrates how $\underline{e}_R$ forms the hypotenuse and $\underline{e}_F$ forms a leg of a right triangle.

The missing[2] leg is the vector $\hat{\underline{Y}}_{Full} - E(\underline{Y})$ and it is this right triangle that elucidates the orthogonality which implies the independence of the Chi

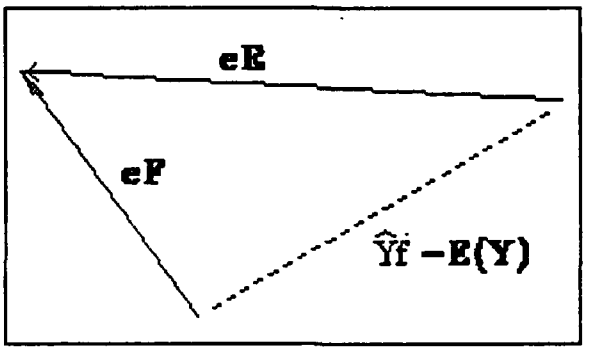

Figure 14. Data Set 1

Square statistics involving their squared lengths. Such independence allows the F statistic to be constructed. The orthogonal relationship of these two vectors, graphically demonstrated, constitutes a rigorous proof of independence which is infinitely more comprehensible to the mathematically naive student than a calculus-based proof. The ratio between the length of the missing triangle leg of Figure 14, $|\hat{\underline{Y}} - \underline{E}(Y)|^2$ and the length of $|\underline{e}_F|^2$ is the core relationship involved in the construction of the F statistic which is computed and displayed by the PPME.

The dimension of the Estimation Space is readily evident through this visualization process. With a Sample Space of n=3 the estimate of the

[2] The dotted line is not produced by the PPME. It is included here to emphasize the conceptual presence of the vector $\hat{\underline{Y}}_{Full} - E(\underline{Y})$.

E(Y) vector always lies on the [1,1,1] vector or on the [1,1] vector when n=2. By visualizing this, the student easily transitions into hyper-dimensions where E(Y) falls on the $[1,1,...,1_n]$ vector. Algebraically, the P value is computed in order to assess the statistical significance of the model fit but is geometrically self evident. By observing the length of $|\hat{Y} - E(Y)|^2$ versus the length of $|e_F|^2$ the statistical significance, or justification for rejecting the null hypothesis can be visually verified. This is exactly what should be facilitated by a constructive learning environment.

The ratio SSR/SSTO, which is displayed by the PPME as SSR, provides a numerical equivalent of what was just demonstrated visually and can be used by the student to see whether or not the Reduced Model can be rejected in favor of the Full Model.

Data set 3 [2,4,6] is the second data set under analysis and it is quite similar to data set 1. The predominant difference is in the value of their means. The graphic depiction of this data set is shown in Figure 15. Compared to the previous data set, the primary difference appears to be the existence of E(Y) and $\hat{Y}$ as non-zero vectors. Graphically, the orthogonality between e_F and $\hat{Y}$- E(Y) is even more evident than with the previous data set.

The PPME verifies the dimensionality of the Estimation Space because $\hat{Y}$ is collinear with the Estimation Space, [1,1,1]. By comparing the

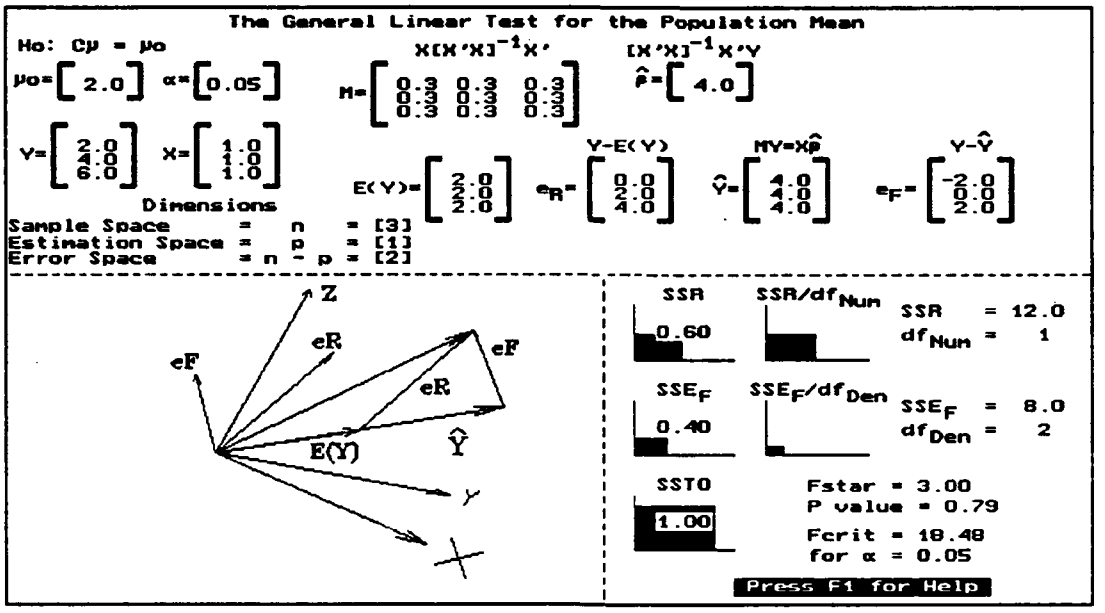

Figure 15. Data Set 3, $\mu_0=2$

E(Y) vector to the $\hat{Y}$ vector the user gains a visceral feeling for the fit of the data. As the angle between E(Y) and $\hat{Y}$ decreases, their correlation increases. This is illustrated by the PPME through the plots of the SSR/SSTO and SSE_F/SSTO; that is, as the explanatory power of the data increases, so does the value of the ratio SSR/SSTO.

This scenario typifies a constructive learning process in which a student can observe and experiment actively with the GLM. Such dynamic experimentation, along with competent guidance, encourages a student to build on previous mathematical foundations, and hence, truly understand the significance of the GLM.

Linear Simple Regression to Estimate E(Y | X)

When the General Linear Model is used to conduct a Simple Linear Regression to estimate E(Y | X) and test about the slope parameter β_1 we assume that its estimator, $\hat{\beta}_1$, is normally distributed. As in the previous scenario, the concepts motivated by this General Linear Test require an understanding of hypothesis testing and the Full and Reduced Models. This scenario uses data set 3 the Design Matrix X_2:

$$X_2 = \begin{bmatrix} 1 & 2 \\ 1 & 3 \\ 1 & 10 \end{bmatrix}$$

The Reduced Model is associated with the null hypothesis, H_0, and is represented as shown:

$$Y = \underline{1}[\beta_0] + \varepsilon$$

where β_0 is unknown. The error for the Reduced Model is given as the difference between the Y_{Red} and the $\hat{Y}_{Red}$ and is written as:

$$e_R = Y_{Red} - \hat{Y}_{Red}$$

where $Y_{Red} = Y - X_{(i,2)}\beta_{10}$ and $\hat{Y}_{Red} = X\beta = \underline{1}\beta_0$.

The Full Model is associated with the alternate hypothesis, H_a, and is represented as shown:

$$Y = X\underline{\beta} + \varepsilon$$

If we estimate ß with $\underline{\hat{\beta}}$, the error for the Full Model is given as the difference between the Y_{Full} and the $\hat{Y}_{Full}$ and is written as:

$$e_R = Y_{Full} - \hat{Y}_{Full}$$

The third scenario uses the PPME to test the slope parameter where $\beta_1 = \beta_{10}$. For this test, data set 3 [2,4,6] is used in a single hypothesis test of $\beta_1 = 0$ using the values of X_2 as specified in Chapter 3 and as shown:

$$X_2 = \begin{bmatrix} 1 & 2 \\ 1 & 3 \\ 1 & 10 \end{bmatrix} .$$

By looking at Figure 16 the student can see the error vectors and by comparing the magnitudes of the error vectors for the Full Model, $\underline{e}_F$ and the Reduced Model, $\underline{e}_R$, he can decide whether or not to reject the null hypothesis.

The orthogonality between the $\underline{e}_F$ and $\underline{\hat{Y}}$ was shown in the previous scenario also applies in this scenario and is exemplified in Figure 17.

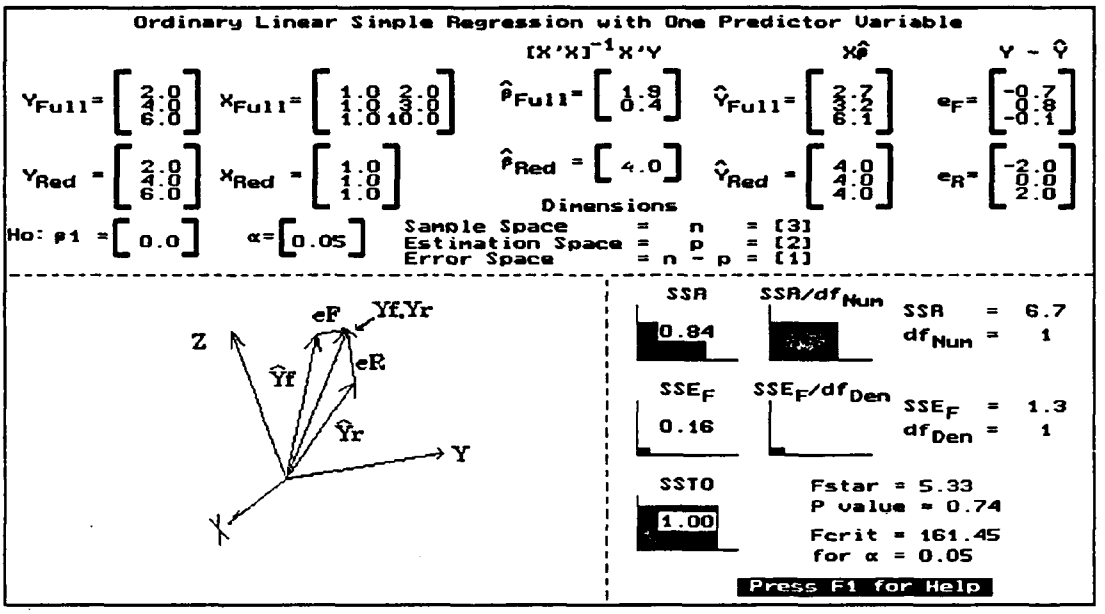

Figure 16. Data Set 3, $\beta_1=0$

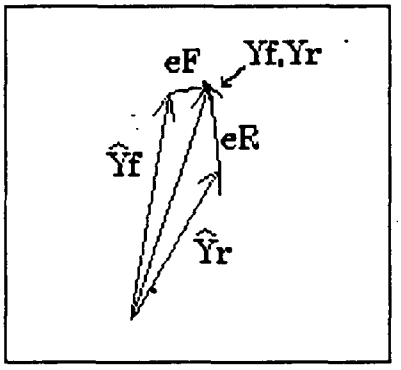

Figure 17. Data Set 3

Figure 16 shows the results of the test of $\beta_1=0$. It also shows the orthogonality between $\underline{e}_F$ and $\underline{\hat{Y}}_R - \underline{\hat{Y}}_F$. From the Pythagorean Theorem the relationship between the vectors in Figure 16 can be defined as shown:

$$|\underline{e}_R|^2 = |\underline{e}_F|^2 + |\underline{\hat{Y}}_F - \underline{\hat{Y}}_R|^2$$

where $|\underline{e}_R|^2 - |\underline{e}_F|^2$ is a Chi Square statistic with degrees of freedom $df_R - df_F$. $|\underline{e}_F|^2$ is also a Chi Square statistic with df_F degrees of freedom.

The Estimation Space for the second scenario was one-dimensional, which resulted in the estimator $\underline{\hat{Y}}$ being restricted to one-dimension along the [1,1,1] vector. The Estimation Space for this scenario is the plane defined by the two column vectors of X_2. Thus, the estimators, $\underline{\hat{Y}}_{Reduced}$ and $\underline{\hat{Y}}_{Full}$ must always lie on that plane as shown in Figure 18.

Notice how easily the PPME illustrates this concept. The PPME encourages

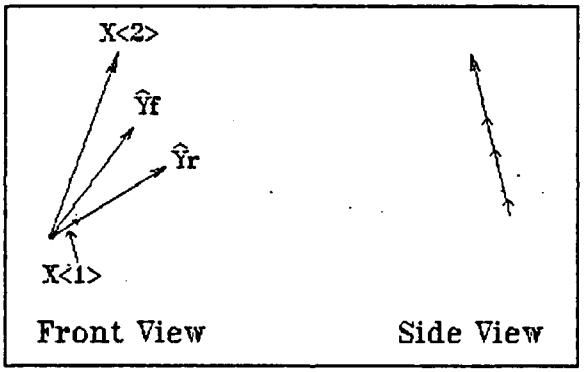

Figure 18. Data Set 3, $ß_{10}=0$

the student to link his current learning experiences to previously mastered concepts and, hence, to understand the GLM as a whole instead of in a shattered and piecemeal fashion. It does this by relating the current General Linear Test to the previous one. Notice how $\underline{\hat{Y}}_{Reduced}$ lies on X<1>[3]. In

[3] X<i> refers to the ith column of the matrix X.

scenario two the Design Matrix was X<1> and the visualization implies that for higher dimensions, the dimension of the Estimation Space will always be the number of columns of the Design Matrix X.

The vectors of key interest for determining the explanatory power of the model are $\underline{e}_F$ and $\underline{e}_R$. The F Statistic, which is calculated by squaring these vectors and dividing them by their degrees of freedom, is used to evaluate the data to determine whether or not the Reduced Model should be rejected.

The PPME graphically demonstrates how different subsets of the GLM relate to the whole. Through the VVAM governance, and the PPME, the student is able to truly construct new personal knowledge. The real-time interaction between the PPME, student and teacher makes the subject come alive for the student.

All three scenarios confirm that the PPME, when used under the guidance of a competent instructor and governance of the VVAM, can encourage students to seek and understand more about the General Linear Model than ever could occur in a passive learning environment and via rote memorization. As a result, the three research hypotheses proposed in Chapter 1 are accepted. Chapter 5 presents the final conclusions of this thesis and recommendations for further research.

V. Conclusions and Recommendations

Final Conclusions

The research of this thesis relied heavily on the VVAM protocol developed by Captain Stone Hansard. Its purpose was to determine if the VVAM could be used to govern a computer-based graphically-supported projective approach to teaching and learning the theory of the General Linear Model. The central hypotheses this research effort sought to validate were

1) such a system could be built
and 2) such a system, once built, would facilitate a constructive, meaningful, lively and real world focused mode of learning of the GLM.

Chapter 3 proposed three scenarios that were analyzed in Chapter 4, scenarios that were specifically designed to assess the validity of the three research hypothesis presented in Chapter 1.

Once the Pearce Projective Modeling Environment (PPME) had been developed, several data sets were employed to test the PPME's efficacy under the three estimation/testing scenarios. The first scenario required the least squares estimation of the Sample Mean and Variance. Five data sets were employed, and the PPME's response to each of them was compared and contrasted. The second scenario, in which a General Linear Test of the Population Mean was made, exercised the PPME using two data sets. The final scenario evaluated the PPME using one data set and a specially selected

Deign Matrix to conduct a Simple Linear Regression Analysis and test of the Slope Coefficient.

In the final analysis, each scenario lead to a firm confirmation of the PPME's ability, under VVAM governance, to foster a constructive mode of learning. Given four criteria, Constructiveness, Meaningfulness, Livingness, and Relatedness to evaluate the results of each experiment, actual evaluation exercises focused on measuring Constructiveness, leaving full scale evaluation of the other three criteria to some future research effort carefully designed to conduct a more formal statistical assessment of the PPME's ability to orchestrate constructive learning activities.

Recommendations for Future Research

Six data sets were used in the evaluation of the PPME. While these data sets are fairly representative of real world system responses, and quite varied, a future research effort could study a broader set of responses and assess their impact on the PPME. Exploration could be made within a hyper-dimensional vector space with the support of MathCAD.

While further evaluation of the PPME, as it stands, would be in order, the PPME programming system, itself, could be modified, and extended, to handle full scale Analysis of Variance (ANOVA). In fact, such research

probably should explore the two main approaches to ANOVA (i.e., means and effect modeling).

Chapter 4 demonstrated how the General Linear Test of a Population Mean can be conceived as a subset of Simple Linear Regression. Future research could explore the ability of the PPME to visually portray subsets of Multiple Linear Regression, and within MLR itself, to portray Step-Wise and All Ways regression analyses. The possibilities are limitless.

In conclusion, this thesis effort was able to demonstrate a graphical system (the PPME) that could be built and supporting experimental evidence obtained to show employment of the PPME does orchestrate a constructive learning environment, under VVAM governance.

As Chapter 2 clearly documents, support for a projective approach to the teaching and learning of the General Linear Model has existed for a very long time. With the availability of cheap and fast personal computers and software to facilitate graphical displays, opportunities to operationalize an even more sophisticated computer-based graphically-supported projective approach to teaching and learning of the General Linear Model are abundant, and most likely will grow exponentially in future years.

Appendix A

Source Code Listing for the PPME

Source Code for the Pearce Projection Modelling Environment was written in Borland's Turbo Pascal V6.0. The Acromolé graphic subroutines were developed by Acrospin, Inc.

```
program PPME;
uses
Crt,Dos,MathMat,GraphMat,TGlobals,Support,Support2,Support3,Mole,Graph;
{
              Copyright 1991 by Stephen D. Pearce
}
{If your are interested in continuing this research, contact Capt Steve
Pearce or Professor Dan Reynolds (at the Air Force Institute of
Technology) for a copy of this source code on disk.}

var
  i,j:              Integer;
  Choice:                Char;
  Output: Text;
  GraphDriver: Integer;
  GraphMode: Integer;
  ErrorCode: Integer;
  msg: String;
  char_H,char_W:Integer;  {Heigth & Width of char in pixels}
  OldStyle: TextSettingsType;

{ Main Program }

{Sample Mean Section}
begin
Assign(Output, 'prn');
Rewrite(Output);

GraphDriver := 3;    {Set Flag to EGA}
GraphMode := 1;      {Set EGA High Mode}
InitGraph(GraphDriver, GraphMode, 'C:Language\TP\Drivers');
ErrorCode := GraphResult;
if ErrorCode <> grOK then
begin
  writeln('Graphics error: ', GraphErrorMsg(ErrorCode));
  Writeln('Program Aborted...');
  Halt(1);
end;

MoleInit;
MoleVideoInit(BackgroundKolor);
MoleClip(Xlo,Ylo,Xhi,Yhi);

char_H:=TextHeight('H');
char_W:=TextWidth('H');
SetBkColor(BackgroundKolor);     {Background Blue}

SetTextJustify(1,1);

SetColor(15); OutTextXY (GetMaxX div 2,char_H,'Welcome to the Pearce
Projective Modeling Environment');
              OutTextXY (GetMaxX div 2,4*char_H,'Which Program would you
like?');
              OutTextXY (GetMaxX div 2, 7*char_H,'Ordinary Least Squares
Regression for the Sample  ean and Variance');
```

```
SetColor(12); OutTextXY (GetMaxX div 2, 7*char_H,'
                         M                    ');
SetColor(15); OutTextXY (GetMaxX div 2, 9*char_H,'General  inear Test
for the Population Mean');
SetColor(12); OutTextXY (GetMaxX div 2, 9*char_H,'        L
                  ');
SetColor(15); OutTextXY (GetMaxX div 2,11*char_H,'Ordinary Linear Simple
egression with One Predictor Variable');
SetColor(12); OutTextXY (GetMaxX div 2,11*char_H,'
R                                      ');
SetColor(15); OutTextXY (GetMaxX div 2, 13*char_H,'E it');
SetColor(12); OutTextXY (GetMaxX div 2, 13*char_H,' x  ');

SetColor(15); OutTextXY (GetMaxX div 2, 16*char_H,'What is Your
Choice?');


repeat
  Choice:= Readkey;
  Choice:=UpCase(Choice);
  OutTextXY (GetMaxX div 2 + 14*char_W, 16*char_H,Choice);
  case Choice of
    'M'     :  ProgramName := 'Mean';
    'L'     :  ProgramName := 'Ltest';
    'R'     :  ProgramName := 'Bivariate';
    'X'     :  ExitRoutine;
    else       OutTextXY (GetMaxX div 2, 18*char_H,'Please Choose M, L,
R, or X.');
  end;
until (Choice = 'M') or (Choice = 'L') or(Choice = 'R') or(Choice =
'X');
ClearDevice;

Repeat
Repeat
  SetVisualPage(0); SetActivePage(0);
  If ProgramName = 'Mean' Then MeanRoutine
  else If ProgramName = 'Ltest' Then LtestRoutine
       else If ProgramName = 'Bivariate' Then BivariateRoutine;

  KeyAction;

  If GetXFlag or GetYFlag or GetBeta1Flag or GetAlphaFlag or GetMuFlag
Then
  While Length(ScreenStack)>0 do Screen(0);

Until (CheckBreakVar.ReturnCode<>AMOkay) Or        (* CTRL BREAK or CTRL
C.   *)
      ((GetKeyboardStatusVar.Down[0] And 2)<>0) Or (* ESC key held down.
    *)
      ((GetKeyboardStatusVar.Missed[0] And 2)<>0); (* ESC key tapped.
    *)

LeaveProgram;
Until False

end.
```

```
Unit TGlobals;

interface
uses MathMat;

(*-------------------------------------------------------------------------
------*)
(* Include object code and Pascal definitons for AcroMole.
    *)
(*-------------------------------------------------------------------------
------*)
{$L MOLE9}            (* Include object code for AcroMole subroutines.
    *)
{ Note:  Most of this subroutine is provided a part of the AcroMole
package}
{AcroMole Variable Definitions}

(*-------------------------------------------------------------------------
------*)
(* Return codes.
    *)
Const AMOkay                    = 0;   AMQueueWaiting        = 11;
      AMInvalidVersion          = 1;   AMSecondsOverflow     = 12;
      AMInvalidRevision         = 2;   AMBreak               = 13;
      AMInvalidAcroMoleSize     = 3;   AMPrintScreen         = 14;
      AMNoVideoModeSet          = 4;   AMPrintScreenError    = 15;
      AMInvalidCameraPosition   = 5;   AMInvalidClippingBits = 16;
      AMUnsupportedVideoMode    = 6;   AMInvalidBuffer       = 17;
      AMUndetectedVideoMode     = 7;   AMVerticalRetrace     = 18;
      AMClippedOutOfWindow      = 8;   AMInvalidScaleFactor  = 19;
      AMQueueFull               = 9;   AMInvalidDOSVersion   = 20;
      AMQueueEmpty              =10;   AMInvalidResetFlag    = 21;
(*-------------------------------------------------------------------------
------*)
(* Divisors for musical notes.
    *)
Const C0  =18243;  C1  =9121;  C2  =4561;  C3  =2280;  C4  =1140;  C5
=570;
      C0s =17219;  C1s =8609;  C2s =4305;  C3s =2152;  C4s =1076;  C5s
=538;
      D0b =17219;  D1b =8609;  D2b =4305;  D3b =2152;  D4b =1076;  D5b
=538;
      D0  =16252;  D1  =8126;  D2  =4063;  D3  =2032;  D4  =1016;  D5
=508;
      D0s =15340;  D1s =7670;  D2s =3835;  D3s =1918;  D4s = 959;  D5s
=479;
      E0b =15340;  E1b =7670;  E2b =3835;  E3b =1918;  E4b = 959;  E5b
=479;
      E0  =14479;  E1  =7240;  E2  =3620;  E3  =1810;  E4  = 905;  E5
=452;
      F0  =13666;  F1  =6833;  F2  =3417;  F3  =1708;  F4  = 854;  F5
=427;
      F0s =12899;  F1s =6450;  F2s =3225;  F3s =1612;  F4s = 806;  F5s
=403;
      G0b =12899;  G1b =6450;  G2b =3225;  G3b =1612;  G4b = 806;  G5b
=403;
      G0  =12175;  G1  =6088;  G2  =3044;  G3  =1522;  G4  = 761;  G5
=380;
      G0s =11492;  G1s =5746;  G2s =2873;  G3s =1437;  G4s = 718;  G5s
=359;
      A0b =11492;  A1b =5746;  A2b =2873;  A3b =1437;  A4b = 718;  A5b
=359;
      A0  =10847;  A1  =5424;  A2  =2712;  A3  =1356;  A4  = 678;  A5
=339;
```

```
      A0s =10238;  A1s =5119;  A2s =2560;  A3s =1280;  A4s = 640;  A5s
=320;
      B0b =10238;  B1b =5119;  B2b =2560;  B3b =1280;  B4b = 640;  B5b
=320;
      B0  = 9664;  B1  =4832;  B2  =2416;  B3  =1208;  B4  = 604;  B5
=302;
(*-------------------------------------------------------------------
------*)
(* Data Blocks types for all of the subroutines except for
BeginAcroMole.     *)
Type CalculateScaleFactorsRecord=Record ReturnCode,Reserved:Word;
                                 ScaleFactorX,ScaleFactorY:Integer;
                                 ResolutionX,ResolutionY,
                                 SizeX,SizeY,
                                 ScaleFactorLo,ScaleFactorHi:Word; End;
     CheckBackgroundSoundRecord=Record ReturnCode:Word; End;
     CheckBreakRecord=Record ReturnCode:Word; End;
     CheckForegroundSoundRecord=Record ReturnCode:Word; End;
     CheckPrintScreenRecord=Record ReturnCode:Word; End;
     CheckVerticalRetraceRecord=Record ReturnCode:Word; End;
     ClearBackgroundSoundRecord=Record ReturnCode:Word; End;
     ClearForegroundSoundRecord=Record ReturnCode:Word; End;
     ClearKeyboardBufferRecord=Record ReturnCode:Word; End;
     Clip2DLineRecord=Record ReturnCode:Word;
                      ScreenX1,ScreenY1,ScreenX2,ScreenY2:Integer;
                      ClippingBits1,ClippingBits2:Byte; End;
     Clip3DLineRecord=Record ReturnCode:Word;
                      ScreenX1,ScreenY1,ScreenX2,ScreenY2:Integer;
                      ClippingBits1,ClippingBits2:Byte;
                      CameraX1,CameraY1,CameraZ1,
                      CameraX2,CameraY2,CameraZ2:Integer; End;
     ConvertSecondsToTicksRecord=Record ReturnCode,Ignored:Word;
                     SecondsLo,SecondsHi:Word;
                     TicksLo,TicksHi:Word; End;
     ConvertTicksToSecondsRecord=Record ReturnCode,Ignored:Word;
                     TicksLo,TicksHi:Word;
                     SecondsLo,SecondsHi:Word; End;
     DetectVideoModeRecord=Record
ReturnCode,VideoMode,MaximumBuffer:Word; End;
     DrawLineRecord=Record
                     ReturnCode:Word;
                     ScreenX1,ScreenY1,ScreenX2,ScreenY2:Integer;
                     Color:Word; End;
     DrawPointRecord=Record
                      ReturnCode:Word;
                      ScreenX,ScreenY:Integer;
                      Color:Word; End;
     DrawRectangleRecord=Record
                     ReturnCode:Word;
                     ScreenX1,ScreenY1,ScreenX2,ScreenY2:Integer;
                     Color:Word; End;
     EndAcroMoleRecord=Record ReturnCode:Word; End;
     GetExecutionTimeRecord=Record
                           ReturnCode,Ignored,SecondsLo,SecondsHi:Word;
End;
     GetKeyboardStatusRecord=Record
                            ReturnCode:Word;
                            Down,Missed:Array[0..7] Of Word; End;
     GetMaximumVideoModeRecord=Record ReturnCode,VideoMode:Word; End;
     GetSuggestedVideoModeRecord=Record ReturnCode,VideoMode:Word; End;
     GetTimeOfDayRecord=Record
                         ReturnCode,Ignored,SecondsLo,SecondsHi:Word;
End;
     GetVideoModeInfoRecord=Record
```

```
                              ReturnCode,VideoMode,BIOSMode:Word;
                              MinimumScreenX,MinimumScreenY,
                              MaximumScreenX,MaximumScreenY:Integer;
                              MaximumColor,MaximumBuffer:Word; End;
     PrintScreenRecord=Record ReturnCode:Word; End;
     QueueBackgroundSoundRecord=Record ReturnCode,Divisor,Ticks:Word;
End;
     QueueForegroundSoundRecord=Record ReturnCode,Divisor,Ticks:Word;
End;
     RestoreOriginalVideoModeRecord=Record ReturnCode:Word; End;
     Set3DCameraRecord=Record
                         ReturnCode:Word;
                         WorldX,WorldY,WorldZ,
                         DirectionX,DirectionY,DirectionZ,
                         UpX,UpY,UpZ,
                         ScaleFactorX,ScaleFactorY,Perspective:Integer;
                         End;
     SetDisplayedBufferRecord=Record ReturnCode,Buffer:Word; End;
     SetDrawingBufferRecord=Record ReturnCode,Buffer:Word; End;
     SetVideoModeRecord=Record
                          ReturnCode:Word;
                          VideoMode:Word;
                          Reset:Word; End;
     SetWindowRecord=Record
                       ReturnCode:Word;

MinimumFilmX,MinimumFilmY,MaximumFilmX,MaximumFilmY:Integer;
                       MinimumScreenX,MinimumScreenY:Integer;
                       End;
     StartImmediateSoundRecord=Record ReturnCode,Divisor:Word; End;
     StopImmediateSoundRecord=Record ReturnCode:Word; End;
     Transform3DEndpointRecord=Record
                                 ReturnCode:Word;
                                 ScreenX,ScreenY,
                                 WorldX,WorldY,WorldZ,
                                 CameraX,CameraY,CameraZ:Integer;
                                 ClippingBits:Byte; End;
     Transform3DPointRecord=Record
                              ReturnCode:Word;
                              ScreenX,ScreenY,
                              WorldX,WorldY,WorldZ:Integer; End;
(*-------------------------------------------------------------------------
------*)
(* AcroMole Address Block type.
   *)
AcroMoleAddressBlockRecord=Record
    CalculateScaleFactors   :Procedure(Var
Data:CalculateScaleFactorsRecord);
    CheckBackgroundSound    :Procedure(Var
Data:CheckBackgroundSoundRecord);
    CheckBreak              :Procedure(Var Data:CheckBreakRecord);
    CheckForegroundSound    :Procedure(Var
Data:CheckForegroundSoundRecord);
    CheckPrintScreen        :Procedure(Var Data:CheckPrintScreenRecord);
    CheckVerticalRetrace    :Procedure(Var
Data:CheckVerticalRetraceRecord);
    ClearBackgroundSound    :Procedure(Var
Data:ClearBackgroundSoundRecord);
    ClearForegroundSound    :Procedure(Var
Data:ClearForegroundSoundRecord);
    ClearKeyboardBuffer     :Procedure(Var
Data:ClearKeyboardBufferRecord);
    Clip2DLine              :Procedure(Var Data:Clip2DLineRecord);
    Clip3DLine              :Procedure(Var Data:Clip3DLineRecord);
```

```
    ConvertSecondsToTicks    :Procedure(Var
Data:ConvertSecondsToTicksRecord);
    ConvertTicksToSeconds    :Procedure(Var
Data:ConvertTicksToSecondsRecord);
    DetectVideoMode          :Procedure(Var Data:DetectVideoModeRecord);
    DrawLine                 :Procedure(Var Data:DrawLineRecord);
    DrawPoint                :Procedure(Var Data:DrawPointRecord);
    DrawRectangle            :Procedure(Var Data:DrawRectangleRecord);
    EndAcroMole              :Procedure(Var Data:EndAcroMoleRecord);
    GetExecutionTime         :Procedure(Var Data:GetExecutionTimeRecord);
    GetKeyboardStatus        :Procedure(Var
Data:GetKeyboardStatusRecord);
    GetMaximumVideoMode      :Procedure(Var
Data:GetMaximumVideoModeRecord);
    GetSuggestedVideomode    :Procedure(Var
Data:GetSuggestedVideoModeRecord);
    GetTimeOfDay             :Procedure(Var Data:GetTimeOfDayRecord);
    GetVideoModeInfo         :Procedure(Var Data:GetVideoModeInfoRecord);
    PrintScreen              :Procedure(Var Data:PrintScreenRecord);
    QueueBackgroundSound     :Procedure(Var
Data:QueueBackgroundSoundRecord);
    QueueForegroundSound     :Procedure(Var
Data:QueueForegroundSoundRecord);
    RestoreOriginalVideoMode:Procedure(Var
Data:RestoreOriginalVideoModeRecord);
    Set3DCamera              :Procedure(Var Data:Set3DCameraRecord);
    SetDisplayedBuffer       :Procedure(Var
Data:SetDisplayedBufferRecord);
    SetDrawingBuffer         :Procedure(Var Data:SetDrawingBufferRecord);
    SetVideoMode             :Procedure(Var Data:SetVideoModeRecord);
    SetWindow                :Procedure(Var Data:SetWindowRecord);
    StartImmediateSound      :Procedure(Var
Data:StartImmediateSoundRecord);
    StopImmediateSound       :Procedure(Var
Data:StopImmediateSoundRecord);
    Transform3DEndpoint      :Procedure(Var
Data:Transform3DEndpointRecord);
    Transform3DPoint         :Procedure(Var Data:Transform3DPointRecord);
End;
(*-------------------------------------------------------------------------
------*)
(* Data Block type for BeginAcroMole.
    *)
BeginAcroMoleRecord=Record
                    ReturnCode:Word;
                    Revision:Word;
                    Version:Word;
                    AcroMoleSize:Word;
                    AcroMolePointer:^AcroMoleAddressBlockRecord; End;

(*-------------------------------------------------------------------------
------*)
(* Procedure definition for BeginAcroMole.
    *)
{$F+} Procedure BeginAcroMole(Var Data:BeginAcroMoleRecord); {$F-}
(*-------------------------------------------------------------------------
------*)


(*-------------------------------------------------------------------------
------*)
(* Constants.
    *)
```

```
(*-------------------------------------------------------------------
------*)
Const
   Xlo:Real=0.0;                (* Window Boundry *)
   Xhi:Real=0.55;               (* Window Boundry *)
   Ylo:Real=0.0;                (* Window Boundry *)
   Yhi:Real=0.55;               (* Window Boundry *)
   Increment=0.05;                        {Angle Change}
   NE=80;
   NL=60;
   FirstPoint:Integer=30;
   FirstLine:Integer=29;

   BackgroundKolor:Word=1;     {Blue}

   MuColor:Word=15;
   AlphaColor:Word=15;
   ObservationColor:Word=4;     {Red}
   Observation2Color:Word=12;    {Light Red}

   ModelSpaceColor:Word=14;     {Yellow}
   ModelSpace2Color:Word=6;     {Brown}

   ErrorColor:Word=13;          {Light Magenta}
   ErrReducedColor:Word=9;      {Light Blue}
   YhatColor:Word=3;            {Cyan}
   MatForeground:Word=7;        {Light Gray}
   SSTOColor:Word=2;            {Green}
   MuVColor:Word=12;
   eRColor:Word=9;
   YhatRedColor:Word=10;
   YhatFullColor:Word=11;

   LabelColor:Word=7;

   (* These override colors produce a black & white version of the
program *)


   ScaleSize:Real=3000;         {Scale used to size arrows}
   ToggleDelay:Real=0.75;       {Time to wait before toggling the Cube
on/off}

(*-------------------------------------------------------------------
------*)
(* Initialized variables.
    *)
(*-------------------------------------------------------------------
------*)
   VideoModeFlag:Boolean=True;  (* Always start by setting video mode.
    *)
   ScreenSize:LongInt=16383;    (* Always start with screen size ratio
of 4:3.*)
   DeltaVideoMode:Integer=0;    (* Always start with suggested video
mode.    *)
   RotMat:Array[0..2,0..2] of Real=
    ((24945, -6782, -20124),
     ( 712, 31294, -9664),       {Rotation Matrix, Initial Camera
Position}
      (21224, 6921, 23976));
   LayerScreen:Word=65535;           {0 = All Off}
                                {1 = Cube On}
                                {2 = Axis On}
```

```
                                    {3 = Cube and Axis On}
                                    {4 = Vectors On}
                                    {7 = All On}
   CubeFlipTime:Real=0;             {Time the cube was last toggled}
   AxisFlipTime:Real=0;
   XFlipTime:Real=0; YFlipTime:Real=0; EFlipTime:Real=0;
FFlipTime:Real=0;
   RFlipTime:Real=0; HFlipTime:Real=0; JFlipTime:Real=0;
ZFlipTime:Real=0;
   UFlipTime:Real=0;
   GetXFlag:Boolean=False; GetYFlag:Boolean=False;
GetBetaIFlag:Boolean=False;
   GetMuFlag:Boolean=False; GetAlphaFlag:Boolean=False;


   HelpScreenOn:Boolean=False;
   ToggleKeyScreenOn:Boolean=False;
   RegressionScreenOn:Boolean=False;
   MScreenOn:Boolean=False;
   FirstRunFlag:Boolean=True;

   ScreenStack:String='';

(*-------------------------------------------------------------------------
------*)
(* Uninitialized variables.
    *)
(*-------------------------------------------------------------------------
------*)
Var          (* These are the data blocks for the AcroMole subroutines
used. *)
   AcroMole                    :AcroMoleAddressBlockRecord;
   BeginAcroMoleVar            :BeginAcroMoleRecord;
   CalculateScaleFactorsVar    :CalculateScaleFactorsRecord;
   CheckBreakVar               :CheckBreakRecord;
   CheckVerticalRetraceVar     :CheckVerticalRetraceRecord;
   ClearKeyboardBufferVar      :ClearKeyboardBufferRecord;
   Clip3DLineVar               :Clip3DLineRecord;
   DetectVideoModeVar          :DetectVideoModeRecord;
   DrawRectangleVar            :DrawRectangleRecord;
   EndAcroMoleVar              :EndAcroMoleRecord;
   GetExecutionTimeVar         :GetExecutionTimeRecord;
   GetKeyboardStatusVar        :GetKeyboardStatusRecord;
   GetMaximumVideoModeVar      :GetMaximumVideoModeRecord;
   GetSuggestedVideoModeVar    :GetSuggestedVideoModeRecord;
   GetVideoModeInfoVar         :GetVideoModeInfoRecord;
   RestoreOriginalVideoModeVar:RestoreOriginalVideoModeRecord;
   Set3DCameraVar              :Set3DCameraRecord;
   SetDisplayedBufferVar       :SetDisplayedBufferRecord;
   SetDrawingBufferVar         :SetDrawingBufferRecord;
   SetVideoModeVar             :SetVideoModeRecord;
   SetWindowVar                :SetWindowRecord;

(*-------------------------------------------------------------------------
---*)
   (* These arrays are for the points and lines actually drawn on the
screen. *)

(*-------------------------------------------------------------------------
---*)
   ScreenLine:Array[0..2,0..NL {(NumberOfLines-1)}] Of Record
          DrawLineVar:DrawLineRecord; End;
```

```
(*-------------------------------------------------------------------------
---*)
   (* These arrays describe the object in the world coordinate system.
    *)

(*-------------------------------------------------------------------------
---*)
   WorldEndpoint:Array[0..NE{(NumberOfEndpoints-1)}] Of Record
          Transform3DEndpointVar:Transform3DEndpointRecord; End;
   WorldLine:Array[0..NL {(NumberOfLines-1)}]
          Of Record Endpoint1,Endpoint2,LineColor:Word;Layer:Word; End;

(*-------------------------------------------------------------------------
---*)
   (* Miscellaneous uninitialized variables.
    *)

(*-------------------------------------------------------------------------
---*)
   ToDraw,ToErase:Word;           (* Which points and lines to draw or
erase. *)
   PreviousTime,DeltaTime:Word;   (* Previous time and change in time.
    *)
   Radius:Real;                   (* Distance from center of object.
     *)
   MinimumColor:Word;             (* Minimum color to use for lines.
    *)
   Points,Lines:Array[0..2] Of Word; (* How many points to draw or
erase.     *)
   SinAngle,SinPosAngle,CosAngle: Real;
   ScreenWidth,ScreenHeight: Integer;
   NumberOfEndpoints:Integer;  (* Number of endpoints,
     *)
   NumberOfLines:Integer;      (* Number of lines,
    *)
   X,Y,BetaHat,M,Mred,Yhat,Error,Error2,Mu,Mu0,Alpha,eF,eR,eRplusY:
matx;

Betal,Yred,Xred,BhatFull,BhatRed,YhatFull,YhatRed,YhatFulleF,YhatRedeR:
matx;
   Scale: real;  {Used to set scale for vectors}
   Xmax,Ymax,Zmax:Real;  {Max size of object; Used to set scale}
   SSEF,SSER: Real; {SSE Full & Reduced}
   SSR,SSTO: Real;
   SSRF,SSRR: Real;
   dfR, dfF, dfSSR: Integer; {degrees of Freedom}
   Fstar, Fcrit, Pvalue: Real;
   LastPoint,LastLine: Integer;
   MPointer,RegressionPointer,HelpPointer,ToggleKeyPointer: Pointer;
   ProgramName:String;


Implementation

(*-------------------------------------------------------------------------
------*)
(* Procedure definition for BeginAcroMole.
    *)
{$F+} Procedure BeginAcroMole(Var Data:BeginAcroMoleRecord); External;
{$F-}
(*-------------------------------------------------------------------------
------*)
```

```
begin
  NumberOfEndpoints:=22;  (* Number of endpoints,
*)
  NumberOfLines:=17;      (* Number of lines,
*)
  SinPosAngle:=Sin(Increment);
  CosAngle:=Cos(Increment);


(* These override colors produce a black & white version of the program
*)

{  BackgroundKolor:=0;

   MuColor:=15;
   AlphaColor:=15;
   ObservationColor:=15;
   Observation2Color:=15;
   ModelSpaceColor:=15;
   ModelSpace2Color:=15;

   ErrorColor:=15;
   ErrReducedColor:=15;
   YhatColor:=15;
   MatForeground:=15;
   SSTOColor:=15;
   MuVColor:=15;
   eRColor:=15;
   YhatRedColor:=15;
   YhatFullColor:=15;
   LabelColor:=15; }
End.
```

```
Unit Support;

Interface
uses TGlobals,Graph,Crt,MathMat,GraphMat,Dos,Mole,Support2;

function FPercentPoint(p: real; k1,k2: integer): real;

function FProb(f: real; k1,k2: integer): real;

Procedure Help;

Procedure ScreenOff;

Procedure ToggleKeyScreen;

Procedure Colors;

Procedure ChangeColors(var Color: Word);

Procedure ListColors;

Procedure MathRoutines;

Procedure RegressionMath;

Procedure BivariateMath;

Procedure RegressionScreen;

Procedure MScreen;

procedure chart(x1,yhi: Integer;      {Upper Left Location}
                x2,y2: Integer;       {Lower Right Location}
              Percent: Real;          {between 0 and 1}
                Color: Word;          {Color of chart}
                Title: String);       {Title of chart}

function TimeInSeconds: Real;

procedure BarGraphs;

Procedure Screen(ScreenNumber: Integer);

procedure KeyAction;

Implementation

function FPercentPoint(p: real; k1,k2: integer): real;

(*   Calculates the inverse F distribution function based on   *)
(*   k1 and k2 degrees of freedom.                             *)

(*  This function was obtained from the book 'Statistical Computing in
    Pascal by D Cooke, A H Craven and G M Clarke, pages 82-89.
    Published by Edward Arnold, London: 1985
    book obtained at Wright State University
*)


var h1,h2: real; {half degrees of freedom k1,k2}
    LnBeta: real; {Log of complete beta function with parameters h1 and
h2}
    Ratio: real; {Beta ratio}
    x: real; {Inverse Beta ratio}
```

```
function LnGamma(w: real): real;

(*  Calculates the logarithm of the gamma function;   *)
(*  w must be such that 2*w is an integer > 0         *)

var
  a,Sum: real;

begin
  a:= Ln(sqrt(Pi));
  Sum:=0;
  w:= w-1;
  while w>0.0 do
  begin
    sum:= sum + ln(w);
    w:=w-1;
  end;
  if w<0.0 then LnGamma:=sum+a
           else LnGamma:=sum;
end; {Function LnGamma}

function BetaRatio(x,a,b,LnBeta: real): real;

(*  Calculates the incomplete beta function ratio with    *)
(*  parameters a and b.  LnBeta is the logarithm of the   *)
(*  complete beta function with parameters a and b.       *)

const
  error=1.0E-7;

var
  c: real;  {c=a+b}
  Factor1,Factor2,Factor3: real; {factors multiplying terms in series}
  i,j: integer; {counters}
  sum: real; {current sum of series}
  Temp: real;
  Term: real;
  xLow: boolean; {status of x which determines the end from which series
                  is evaluated}
  y: real; {adjusted argument}

begin
  if (x=0) or (x=1) then sum := x
  else begin
    c:= a+b;
    if a<c*x then begin
      xLow:=true;
      y:=x;
      x:=1-x;
      Temp:=a;
      a:=b;
      b:=Temp; end
    else begin
      xLow:=false;
      y:=1-x; end;
    Term:=1;
    j:=0;
    Sum:=1;
    i:=trunc(b+c*y)+1;
    Factor1:=x/y;
    repeat
      j:=j+1;
      i:=i-1;
      if i>=0 then begin
```

```
        Factor2:=b-j;
        if i=0 then Factor2:=x;
      end;
      Term:=Term*Factor2*Factor1/(a+j);
      Sum:=Sum+Term;
    until (abs(Term) <=Sum) and (abs(Term) <= error*sum);
    Factor3:=exp(a*ln(x)+(b-1)*ln(y)-LnBeta);
    Sum:=Sum*factor3/a;
    If xLow then Sum:=1-Sum;
  end;
  BetaRatio:=Sum;
end; {function BetaRatio}

function InverseBetaRatio(Ratio,a,b,LnBeta: real): real;

(*  Calculates the inverse of the incomplete beta function ratio    *)
(*  with parameters a and b.  LnBeta is the logarithm of the        *)
(*  complete beta function with parameters a and b.                 *)

const
  error=1.0E-7;

var
  c: real; {c=a+b}
  LargeRatio: boolean;
  temp1,temp2,temp3,temp4: real;
  x,x1: real; {successive estimates of inverse ratio}
  y: real; {adjustment during Newton iteration}

begin
  if (ratio=0) or (ratio=1) then x:=Ratio
  else begin
    LargeRatio:=false;
    if Ratio>0.5 then begin
      LargeRatio:=true;
      Ratio:=1-Ratio;
      temp1:=a;
      b:=a;
      a:=temp1; end;
    c:=a+b;
    (* claculates initial estimate for x *)
    if Ratio<=0 then begin
      writeln(output,' Ratio = ',Ratio);
      writeln(output,' This is too small or negative.'); Halt; end;
    temp1:= sqrt(-ln(Ratio*Ratio));
    temp2:= 1.0+temp1*(0.99229 + 0.04481*temp1);
    temp2:= temp1-(2.30753 + 0.27061*temp1)/temp2;
    if (a>1) and (b>1) then begin
      temp1:= (temp2*temp2-3.0)/6.0;
      temp3:= 1.0/(a+a-1.0);
      temp4:= 1.0/(b+b-1.0);
      x1:= 2.0/(temp3+temp4);
      x:= temp1+5.0/6.0-2.0/(3.0*x1);
      x:= temp2*sqrt(x1+temp1)/x1-x*(temp4-temp3);
      x:= a/(a+b*exp(x+x)); end
    else begin
      temp1:= b+b;
      temp3:=1.0/(9.0*b);
      temp3:=1.0-temp3+temp2*sqrt(temp3);
      temp3:= temp1*temp3*temp3*temp3;
      if temp3>0 then begin
        temp3:=(4.0*a+temp1-2.0)/temp3;
        if temp3>1 then x:=1.0-2.0/(1+temp3)
        else x:= exp((ln(Ratio*a)+LnBeta)/a); end
```

```
      else x:=1.0-exp((ln((1-Ratio)*b)+LnBeta)/b);
    end;

    (*  Newton Iteration  *)
    repeat
      y:=BetaRatio(x,a,b,LnBeta);
      y:= (y-Ratio)*exp((1-a)*Ln(x)+(1-b)*Ln(1-x)+LnBeta);
      temp4:= y;
      x1:= x-y;
      while (x1 <= 0) or (x1 >=1) do begin
        temp4:=temp4/2;
        x1:= x-temp4; end;
      x:= x1;
    until abs(y) < error;
    if LargeRatio then x:=1-x;
  end;
  InverseBetaRatio:=x;
end;  {Function InverseBetaRatio}

begin  {FPercentPoint}
  h1:= 0.5*k2;
  h2:= 0.5*k1;
  Ratio:= 1-p;
  LnBeta:=LnGamma(h1) + LnGamma(h2) - LnGamma(h1+h2);
  x:= InverseBetaRatio(Ratio,h1,h2,LnBeta);
  FPercentPoint:= k2*(1-x)/(k1*x);
end;  {Function FPercentPoint}


function FProb(f: real; k1,k2: integer): real;

(*  The distribution function of the F distribution based on k1 and k2
*)
(*  degrees of freedom;
*)

(*  This function was obtained from the book 'Statistical Computing in
    Pascal by D Cooke, A H Craven and G M Clarke, pages 82-89.
    Published by Edward Arnold, London: 1985
    book obtained at Wright State University
*)

var
  h1,h2: real;  {modified degrees of freedom}
  LnBeta: real; {Log of complete beta function with parmaters h1 & h2}
  x: real;  {argument of incomplete beta function}

function LnGamma(w: real): real;

(*  Calculates the logarithm of the gamma function;  *)
(*  w must be such that 2*w is an integer > 0        *)

var
  a,Sum: real;

begin
  a:= Ln(sqrt(Pi));
  Sum:=0;
  w:= w-1;
  while w>0.0 do
  begin
    sum:= sum + ln(w);
    w:=w-1;
```

```
  end;
  if w<0.0 then LnGamma:=sum+a
           else LnGamma:=sum;
end; {Function LnGamma}

function BetaRatio(x,a,b,LnBeta: real): real;

(*  Calculates the incomplete beta function ratio with   *)
(*  parameters a and b.  LnBeta is the logarithm of the  *)
(*  complete beta function with parameters a and b.      *)

const
  error=1.0E-7;

var
  c: real;  {c=a+b}
  Factor1,Factor2,Factor3: real; {factors multiplying terms in series}
  i,j: integer; {counters}
  sum: real; {current sum of series}
  Temp: real;
  Term: real;
  xLow: boolean; {status of x which determines the end from which series
                  is evaluated}
  y: real; {adjusted argument}

begin
  if (x=0) or (x=1) then sum := x
  else begin
    c:= a+b;
    if a<c*x then begin
      xLow:=true;
      y:=x;
      x:=1-x;
      Temp:=a;
      a:=b;
      b:=Temp; end
    else begin
      xLow:=false;
      y:=1-x; end;
    Term:=1;
    j:=0;
    Sum:=1;
    i:=trunc(b+c*y)+1;
    Factor1:=x/y;
    repeat
      j:=j+1;
      i:=i-1;
      if i>=0 then begin
        Factor2:=b-j;
        if i=0 then Factor2:=x;
      end;
      Term:=Term*Factor2*Factor1/(a+j);
      Sum:=Sum+Term;
    until (abs(Term) <=Sum) and (abs(Term) <= error*sum);
    Factor3:=exp(a*ln(x)+(b-1)*ln(y)-LnBeta);
    Sum:=Sum*factor3/a;
    If xLow then Sum:=1-Sum;
  end;
  BetaRatio:=Sum;
end; {function BetaRatio}

begin {main function}
  h1:= 0.5*k1;
  h2:= 0.5*k2;
```

```
  x:= h2/(h2+h1*f);
  LnBeta:= LnGamma(h1)+LnGamma(h2)-LnGamma(h1+h2);
  FProb:= 1-BetaRatio(x,h2,h1,LnBeta);
end; {Function FProb}


Procedure Help;

const
  X1:Integer=360;
  Y1:Integer=158;
  X2:Integer=639;
  Y2:Integer=349;
  col:Integer=25;

var
  i: Integer;
  Size: Word;
  ViewPort: ViewPortType;
  OldStyle: TextSettingsType;

begin
 If Not HelpScreenOn Then begin          {If the Help Screen is not}
  GetTextSettings(OldStyle);             {Displayed then Display it}
  SetVisualPage(0);
  SetActivePage(1);
  Size := ImageSize(X1,Y1,X2,Y2);
  If Size>MemAvail Then writeln(output,'Size=',Size,'
MemAvail=',MemAvail);
  Mark(HelpPointer);
  GetMem(HelpPointer, Size);
  GetImage(X1,Y1,X2,Y2,HelpPointer^);
  For i:= 1 to 2 do begin
    SetViewPort(X1,Y1,X2,Y2,True);
    GetViewSettings(ViewPort);
    SetFillStyle(1,15);
    Bar(0,0,X2-X1,Y2-Y1);                {Clear ViewPort}
    SetFillStyle(1,7);
    Bar(5,4,X2-X1-5,Y2-Y1-4);
    SetColor(0);
    SetTextJustify(1,1);
    OutTextXY((X2-X1) div 2,10,'Help');
    SetTextJustify(0,1);
    MoveTo(col,GetY+20);
    OutText('F1     Toggle the Help Screen'); MoveTo(col,GetY+10);
    OutText('F2     Vector Toggling Info'); MoveTo(col,GetY+10);
    If (ProgramName = 'Ltest') or (ProgramName = 'Bivariate') Then Begin
      OutText('F3     Toggle Regression Info'); MoveTo(col,GetY+10);
end;
    If ProgramName = 'Ltest' Then Begin
      OutText('F5     Modify '#230'o'); MoveTo(col,GetY+10); end;
    If ProgramName = 'Bivariate' Then Begin
      OutText('F4     Toggle Projection Matrix'); MoveTo(col,GetY+10);
      OutText('F5     Modify '#225'1'); MoveTo(col,GetY+10); end;
    If (ProgramName = 'Ltest') or (ProgramName = 'Bivariate') Then Begin
      OutText('F6     Modify '#224); MoveTo(col,GetY+10); end;
    OutText('F7     Modify Y Matrix'); MoveTo(col,GetY+10);
    OutText('F8     Modify X Matrix'); MoveTo(col,GetY+10);
{   OutText('F9*    Print Vector Array'); MoveTo(col,GetY+10);  }
{   OutText('F10*   Print New Colors'); MoveTo(col,GetY+10);    }
    OutText('F10    Complete Matrix Entry'); MoveTo(col,GetY+15);
    OutText('Home   Return to Initial View'); MoveTo(col,GetY+10);
    OutText('ESCape Exit Program'); MoveTo(col,GetY+10);
    SetTextJustify(1,2); MoveTo((X2-X1) div 2,GetY+10);
```

```
    OutText('Press F1 to Remove Help'); MoveTo((X2-X1) div 2,GetY+15);
{   OutText('* Development Version Only'); }
    MoveTo((X2-X1) div 2,Y2-Y1-15); OutText('Written by Steve Pearce');
    SetColor(15);
    SetVisualPage(1);
    SetActivePage(0);
  end;
    HelpScreenOn:=True;
 end
 else begin                              {If the Help Screen is }
   SetVisualPage(0);                     {Displayed then Erase it}
   SetActivePage(1);
   for i:= 1 to 2 do begin
     SetViewPort(0,0,GetMaxX,GetMaxY,True);
     PutImage(X1,Y1,HelpPointer^,0);     {0=Copy}
     SetVisualPage(1);
     SetActivePage(0);
   end;
   Release(HelpPointer);
   HelpScreenOn:=False;
 end;
end; {Procedure Help}

Procedure ToggleKeyScreen;

const
  X1:Integer=360;
  Y1:Integer=120;
  X2:Integer=639;
  Y2:Integer=349;
  col:Integer=25;

var
  i: Integer;
  Size: Word;
  ViewPort: ViewPortType;
  OldStyle: TextSettingsType;

begin
 If ProgramName = 'Ltest' Then Y1:=200;
 If ProgramName = 'Mean' Then Y1:=200;
 If Not ToggleKeyScreenOn Then begin        {If the Help Screen is not}
   GetTextSettings(OldStyle);              {Displayed then Display it}
   SetViewPort(0,0,GetMaxX,GetMaxY,True);
   SetVisualPage(0);
   SetActivePage(1);
   Size := ImageSize(X1,Y1,X2,Y2);
   If Size>MemAvail Then writeln(output,'Size=',Size,'
MemAvail=',MemAvail);
   Mark(ToggleKeyPointer);
   GetMem(ToggleKeyPointer, Size);
   GetImage(X1,Y1,X2,Y2,ToggleKeyPointer^);
   For i:= 1 to 2 do begin
     SetViewPort(X1,Y1,X2,Y2,True);
     GetViewSettings(ViewPort);
     SetFillStyle(1,15);
     Bar(0,0,X2-X1,Y2-Y1);               {Clear ViewPort}
     SetFillStyle(1,7);
     Bar(5,4,X2-X1-5,Y2-Y1-4);
     SetColor(0);
     SetTextJustify(1,1);
     OutTextXY((X2-X1) div 2,10,'Turning Vectors On/Off');
     SetTextJustify(0,1);
     MoveTo(col,GetY+30);
```

```
      OutText('Alt A    Toggle Axis'); MoveTo(col,GetY+10);
      OutText('Alt C    Toggle Cube'); MoveTo(col,GetY+15);
      If ProgramName = 'Mean' Then Begin
        OutText('Alt E    Toggle Error'); MoveTo(col,GetY+10); end;
      If ProgramName = 'Ltest' Then Begin
        OutText('Alt E    Toggle E(Y) '); MoveTo(col,GetY+10); end;
      If (ProgramName = 'Ltest') or (ProgramName = 'Bivariate') Then
begin
        OutText('Alt F    Toggle e '); MoveTo(col+1,GetY+5);
        OutText('                  F'); MoveTo(col,GetY+10); end;
      OutText('                ^       '); MoveTo(col,GetY+5);
      OutText('Alt H    Toggle Y       ');
      If ProgramName = 'Bivariate' Then Begin
        MoveTo(col,GetY+5);
        OutText('                  Reduced'); MoveTo(col,GetY+10);
        OutText('                ^     '); MoveTo(col,GetY+5);
        OutText('Alt J    Toggle Y     '); MoveTo(col,GetY+5);
        OutText('                  Full');  end;
      MoveTo(col,GetY+10);
      If (ProgramName = 'Ltest') or (ProgramName = 'Bivariate') Then
begin
        OutText('Alt R    Toggle e '); MoveTo(col,GetY+5);
        OutText('                  R'); MoveTo(col,GetY+10); end;
      If ProgramName = 'Bivariate' Then Begin
        OutText('Alt U    Toggle Y       '); MoveTo(col,GetY+5);
        OutText('                  Reduced'); MoveTo(col,GetY+10);
        OutText('Alt X    Toggle Column 1 of X'); MoveTo(col,GetY+15);
end;
      If (ProgramName = 'Mean') or (ProgramName = 'Ltest') Then Begin
        OutText('Alt X    Toggle X     '); MoveTo(col,GetY+10); end;
      OutText('Alt Y    Toggle Y     '); MoveTo(col,GetY+5);
      If ProgramName = 'Bivariate' Then Begin
        OutText('                  full'); MoveTo(col,GetY+10);
        OutText('Alt Z    Toggle Column 2 of X'); MoveTo(col,GetY+10);
end;
      SetTextJustify(1,2); MoveTo((X2-X1) div 2,GetY+10);
      OutText('Press F2 to Remove this Screen'); MoveTo((X2-X1) div
2,GetY+15);
      SetColor(15);
      SetVisualPage(1);
      SetActivePage(0);
    end;
    ToggleKeyScreenOn:=True;
  end
  else begin                              {If the Help Screen is }
    SetVisualPage(0);                     {Displayed then Erase it}
    SetActivePage(1);
    for i:= 1 to 2 do begin
      SetViewPort(0,0,GetMaxX,GetMaxY,True);
      PutImage(X1,Y1,ToggleKeyPointer^,0);    {0=Copy}
      SetVisualPage(1);
      SetActivePage(0);
    end;
    Release(ToggleKeyPointer);
    ToggleKeyScreenOn:=False;
  end;
end;

Procedure ScreenOff;

begin
  {    If BivariateScreenOn then BivariateScreen;    }
   If ToggleKeyScreenOn then ToggleKeyScreen;
   If HelpScreenOn then Help;
```

```
end;

Procedure Colors;

const
  X1:Integer=360;
  Y1:Integer=250;
  X2:Integer=639;
  Y2:Integer=349;
  col:Integer=25;

var
  i: integer;
  P: Pointer;
  Size: Word;
  ViewPort: ViewPortType;
  OldStyle: TextSettingsType;

begin
  GetTextSettings(OldStyle);
  SetVisualPage(0);
  SetActivePage(1);
  SetViewPort(0,0,GetMaxX,GetMaxY,True);
  Size := ImageSize(X1,Y1,X2,Y2);
  If Size>MemAvail Then writeln(output,'Size=',Size,'
MemAvail=',MemAvail);
  Mark(P);
  GetMem(P, Size);
  GetImage(X1,Y1,X2,Y2,P^);
  SetViewPort(X1,Y1,X2,Y2,True);
  GetViewSettings(ViewPort);
  SetFillStyle(1,15);
  Bar(0,0,X2-X1,Y2-Y1);              {Clear ViewPort}
  SetFillStyle(1,1);
  Bar(5,4,X2-X1-5,Y2-Y1-4);
  SetColor(15);
  OutTextXY((X2-X1) div 2,10,'Colors');
  SetTextJustify(0,1);
  MoveTo(col,GetY+20);
  OutText('                1 1 1    '); MoveTo(col,GetY+10);
  OutText('   • 0 2 4 6 8 0 2 4    '); MoveTo(col,GetY+30);

  OutText('      1 3 5 7 9 1 1 1    '); MoveTo(col,GetY+10);
  OutText('                1 3 5    ');

  SetTextJustify(1,2); MoveTo((X2-X1) div 2,GetY+10);
  OutText('Hit Return to Continue'); MoveTo((X2-X1) div 2,GetY+10);

  for i := 0 to 15 do begin
    SetFillStyle(1,i);
    case odd(i) of
      True:  Bar(68+8*i,40,68+8*(i+1)-8,53);   {Draw Bar}
     False:  Bar(68+8*i,35,68+8*(i+1)-8,48);   {Draw Bar}
  end; end;

  SetColor(15);
  SetVisualPage(1);
  repeat
  until KeyPressed;
  SetVisualPage(0);
  SetViewPort(0,0,GetMaxX,GetMaxY,True);
  PutImage(X1,Y1,P^,0);     {0=Copy}
  Release(P);
end;
```

```
Procedure ChangeColors(var Color: Word);

const
  X1:Integer=360;
  Y1:Integer=249;
  X2:Integer=639;
  Y2:Integer=349;
  col:Integer=25;

var
  Key: char;
  Number: String;
  Ok: Boolean;
  i,code: integer;
  P: Pointer;
  msg: String;
  Size,Kolor: Word;
  ViewPort: ViewPortType;
  OldStyle: TextSettingsType;

begin
  GetTextSettings(OldStyle);
  SetVisualPage(0);
  SetActivePage(1);
  SetViewPort(0,0,GetMaxX,GetMaxY,True);
  Size := ImageSize(X1,Y1,X2,Y2);
  If Size>MemAvail Then writeln(output,'Size=',Size,'
MemAvail=',MemAvail);
  Mark(P);
  GetMem(P, Size);
  GetImage(X1,Y1,X2,Y2,P^);
  SetViewPort(X1,Y1,X2,Y2,True);
  GetViewSettings(ViewPort);
  SetFillStyle(1,15);
  Bar(0,0,X2-X1,Y2-Y1);              {Clear ViewPort}
  SetFillStyle(1,1);
  Bar(5,4,X2-X1-5,Y2-Y1-4);
  SetColor(15);

  SetTextJustify(1,1);
  Str(Color:2,msg);
  msg:='Current Color Number is '+msg;
  OutTextXY((X2-X1) div 2,10,msg);
  SetTextJustify(0,1);
  MoveTo(col,GetY+20);

  OutText('                1 1 1    '); MoveTo(col,GetY+10);
  OutText('     0 2 4 6 8 0 2 4    '); MoveTo(col,GetY+30);

  OutText('      1 3 5 7 9 1 1 1   '); MoveTo(col,GetY+10);
  OutText('                1 3 5   ');

  SetTextJustify(0,1); MoveTo(col,GetY+10);
  OutText('Which Color # do you want?');


  for i := 0 to 15 do begin
    SetFillStyle(1,i);
    case odd(i) of
      True:  Bar(64+8*i,40,78+8*i-8,53);   {Draw Bar}
     False:  Bar(64+8*i,35,78+8*i-8,48);   {Draw Bar}
  end; end;

  SetColor(15);
```

```
  SetVisualPage(1);
  Ok := False;
  msg:='';
  repeat
    Key := ReadKey;
    If (Ord(Key)>47) and (Ord(Key)<58) and (length(msg) <2) then begin
      msg:=msg+key;
      OutText(key);
      end
    else if Ord(Key)=13 then begin
      val(msg,Kolor,code);
      If (code = 0) and (Kolor>=0) and (Kolor<=15) then Color:=Kolor
      else OutTextXY(col,GetY+10,'Color NOT Changed'); Delay(750);
      Ok:=True;
    end;

  until Ok;
  SetVisualPage(0);.
  SetViewPort(0,0,GetMaxX,GetMaxY,True);
  PutImage(X1,Y1,P^,0);     {0=Copy}
  Release(P);
end;  {Procedure ChangeColors}

Procedure ListColors;

var
  i: integer;
  Color: String;
procedure PrintColor(i:integer);

begin
  Case i of

   0:     Writeln(output,'Black');
   1:     Writeln(output,'Blue');
   2:     Writeln(output,'Green');
   3:     Writeln(output,'Cyan');
   4:     Writeln(output,'Red');
   5:     Writeln(output,'Magenta');
   6:     Writeln(output,'Brown');
   7:     Writeln(output,'White');
   8:     Writeln(output,'Dark Gray');
   9:     Writeln(output,'Light Blue');
  10:     Writeln(output,'Light Green');
  11:     Writeln(output,'Light Cyan');
  12:     Writeln(output,'Light Red');
  13:     Writeln(output,'Light Magenta');
  14:     Writeln(output,'Yellow');
  15:     Writeln(output,'Intense White');
end;
end; {Procedure PrintColor}


begin
  Write(output,'Alt-X     X[1,i]     ',ModelSpaceColor:2,':');
PrintColor(ModelSpaceColor);
  Write(output,'Alt-Z     X[2,i]     ',ModelSpace2Color:2,':');
PrintColor(ModelSpace2Color);
  Write(output,'Alt-Y     Y full     ',ObservationColor:2,':');
PrintColor(ObservationColor);
  Write(output,'Alt-U     Y red      ',Observation2Color:2,':');
PrintColor(Observation2Color);
  Write(output,'Alt-F      eF        ',ErrorColor:2,':');
PrintColor(ErrorColor);
```

```
  Write(output,'Alt-R      eR         ',ErrReducedColor:2,':');
PrintColor(ErrReducedColor);
  Write(output,'Alt-H     YhatRed     ',YhatRedColor:2,':');
PrintColor(YhatRedColor);
  Write(output,'Alt-J     YhatFull    ',YhatFullColor:2,':');
PrintColor(YhatFullColor);


end;



Procedure MathRoutines;

var
  XP,XPX,XPXI,T1,T2,XI,One,Xtemp,ErrorT:      matx;

begin
  Mat_Transpose(X,XP);
  MatMult(XP,X,XPX);
  MatInvert(XPX,XPXI);
  MatMult(XPXI,XP,T1);
  MatMult(T1,Y,BetaHat);      {BetaHat=[(X'X)^-1]X'Y}
  MatMult(X,T1,M);            {M=X[(X'X)^-1]X'}
  MatMult(M,Y,Yhat);
  MatSub(Y,Yhat,Error);       {Error Vector = Y - Yhat}
  MatAdd(Error,Yhat,Error2); {Error Vector added to Yhat}
end; {Procedure MathRoutines}


Procedure RegressionMath;

var
  SEF,SER,eFT,eRT:      matx;
  n,p: Integer;

begin
  MatSub(Y,Yhat,eF);
  MatSub(Y,Mu,eR);
{  writeln(output,'Y=     ',Y.data[1,1]:4,'  ',Y.data[2,1]:4,'
',Y.data[3,1]:4);
  writeln(output,'Yhat= ',Yhat.data[1,1]:4,'  ',Yhat.data[2,1]:4,'
',Yhat.data[3,1]:4);
  writeln(output,'Mu=   ',Mu.data[1,1]:4,'  ',Mu.data[2,1]:4,'
',Mu.data[3,1]:4);
  writeln(output,'eR=   ',eR.data[1,1]:4,'  ',eR.data[2,1]:4,'
',eR.data[3,1]:4);        }
  Mat_Transpose(eF,eFT); Mat_Transpose(eR,eRT);
  MatMult(eFT,eF,SEF); SSEF:=SEF.data[1,1];
  MatMult(eRT,eR,SER); SSER:=SER.data[1,1];
  SSR:=SSER-SSEF;
  SSTO:=SSER;

  Error := eF;
  MatAdd(Error,Yhat,Error2);      {eF + Yhat}
  MatAdd(eR,Mu,eRplusY);         {eR + E(Y)  [Mu=E(Y)]}
{  writeln(output,'eR+Mu=',eRplusY.data[1,1]:4,'
',eRplusY.data[2,1]:4,'  ',eRplusY.data[3,1]:4); }

  n := X.rows;
  p := X.cols;

  dfR := n - (p-1);
  dfF := n - p;        {df.denominator}
```

```
  dfSSR := dfR-dfF;    {df.numerator}

{Writeln(output,'dfr=',dfr:2,' dff=',dff:2,'  dfssr=',dfssr:2);}
  If (dfSSR=0) or (SSEF=0) or (dfF=0) Then
    writeln(output,'dfSSR=',dfSSR,'  SSEF=',SSEF,'  dfF=',dfF)
   else Fstar := (SSR/dfSSR) / (SSEF/dfF);
  Fcrit:=FPercentPoint((1-Alpha.data[1,1]),dfSSR,dfF);
  Pvalue:= FProb(Fstar,dfSSR,dfF);

end; {Procedure RegressionMath}

Procedure BivariateMath;

var
  i,n,p: Integer;
  X2,X2k,
  eFt,eRt,SEF,SER: matx;

Procedure Bhat(X,Y: matx;
               var Z: matx);
var
  XT,XTX,XTXI,XTXIXT: matx;

begin
  Mat_Transpose(X,XT);
  MatMult(XT,X,XTX);
  MatInvert(XTX,XTXI);
  MatMult(XTXI,XT,XTXIXT);
  MatMult(XTXIXT,Y,Z);
end;

Procedure Mprojection(X: matx;
        var M: matx);
var
  XT,XTX,XTXI,XXTXI: matx;

begin
  Mat_Transpose(X,XT);
  MatMult(XT,X,XTX);
  MatInvert(XTX,XTXI);
  MatMult(X,XTXI,XXTXI);
  MatMult(XXTXI,XT,M);
end;

begin
  for i:= 1 to X.rows do begin
    Xred.data[i,1] := 1;              {Calculate X reduced}
    X2.data[i,1] := X.data[i,2]; end;
  X2.rows:=X.rows; X2.cols:=1;        {X2 is second Column of X}

  Mat_k_Mult(X2,X2k,Betal.data[1,1]);
  MatSub(Y,X2k,Yred);                 {Calculate Y reduced}
  Bhat(X,Y,BhatFull);                 {Calculate BhatFull}
  Bhat(Xred,Yred,BhatRed);            {Calculate BhatReduced}

  MatMult(X,BhatFull,YhatFull);       {Calculate YhatFull}
  MatMult(Xred,BhatRed,YhatRed);      {Calculate YhatRed}

  MatSub(Y,YhatFull,eF);              {Calculate eF}
  MatSub(Yred,YhatRed,eR);            {Calculate eR}

  Mprojection(X,M);                   {Calculate M}
  Mprojection(Xred,Mred);             {Calculate Mreduced}
```

```
  Mat_Transpose(eF,eFT); Mat_Transpose(eR,eRT);
  MatMult(eFT,eF,SEF); SSEF:=SEF.data[1,1];
  MatMult(eRT,eR,SER); SSER:=SER.data[1,1];
  SSR:=SSER-SSEF;
  SSTO:=SSER;

  n := X.rows;
  p := X.cols;

  dfR := n - (p-1);
  dfF := n - p;        {df.denominator}
  dfSSR := dfR-dfF;    {df.numerator}

  MatAdd(YhatFull,eF,YhatFulleF);
  MatAdd(YhatRed,eR,YhatRedeR);

  If (dfSSR=0) or (SSEF=0) or (dfF=0) Then Fstar:=999
  else Fstar := (SSR/dfSSR) / (SSEF/dfF);

  Fcrit:=FPercentPoint((1-Alpha.data[1,1]),dfSSR,dfF);
  Pvalue:= FProb(Fstar,dfSSR,dfF);
end;


Procedure RegressionScreen;

const
  X1:Integer=360;
  Y1:Integer=14;              {Location of Window on Screen}
  X2:Integer=639;
  Y2:Integer=349;
  col:Integer=25;

var
  i: integer;
  Size: Word;
  ViewPort: ViewPortType;
  OldStyle: TextSettingsType;
  msg: string;

begin
  SetVisualPage(0);
  SetActivePage(0);
  GetTextSettings(OldStyle);
  SetTextJustify(1,1);

  SetFillStyle(1,7);
  Bar(430,335,580,345); SetColor(0);
  OutTextXY(505,340,'     Working     ');
  SetColor(15);

  If Not RegressionScreenOn Then begin        {If the Help Screen is
not}
    SetActivePage(1);                         {Displayed then Display
it}
    SetViewPort(0,0,GetMaxX,GetMaxY,True);
    Size := ImageSize(X1,Y1,X2,Y2);
    If Size>MemAvail then writeln(output,'Size=',Size,'
MemAvail=',MemAvail);
    Mark(RegressionPointer);
    GetMem(RegressionPointer, Size);
    GetImage(X1,Y1,X2,Y2,RegressionPointer^);

    For i:= 1 to 2 do begin
```

```
      SetViewPort(X1,Y1,X2,Y2,True);
      GetViewSettings(ViewPort);
      SetFillStyle(1,15);
      Bar(0,0,X2-X1,Y2-Y1);              {Clear ViewPort}
      SetFillStyle(1,7);
      Bar(5,4,X2-X1-5,Y2-Y1-4);
      SetColor(0);
      SetTextJustify(1,1);
      OutTextXY((X2-X1) div 2,10,'Regression Calculations');

      SetTextJustify(0,1);

      If ProgramName = 'Bivariate' Then begin
        MoveTo(col+129,GetY+31); OutText('^');
        MoveTo(col-15,GetY+5);
        OutText('(I-M)Y = e  = Y - Y    ='); MoveTo(col-14,GetY+5);
        OutText('          F        Full ');
      end;

      If ProgramName = 'Ltest' Then Begin
        MoveTo(col,GetY+31);
        OutText('                  ^  '); MoveTo(col,GetY+5);
        OutText('(I-M)Y = e  = Y - Y ='); MoveTo(col,GetY+5);
        OutText('          F          ');
      end;

      If eF.rows = 3 then
GMatWrite(eF,4,1,'',X1+col+177,Y1+GetY-25,ErrorColor,7,0)
                  else
GMatWrite(eF,4,1,'',X1+col+177,Y1+GetY-20,ErrorColor,7,0);
      MoveTo(col,GetY+60);
      OutText('SSE  =  e    =  ');
      Str(SSEF:4:2,msg); OutText(msg);
      MoveTo(col+1,GetY+5); OutText('   F     F');
      SetLineStyle(0,0,1);
      Line(col+60,GetY-9, col+60,GetY+2);
      Line(col+83,GetY-9, col+83,GetY+2);
      OutTextXY(col+87,GetY-8,'2');

      If ProgramName = 'Bivariate' Then begin
        MoveTo(col+25,GetY+25);
        OutText('         ^    '); MoveTo(col+25,GetY+5);
        OutText('e  = Y - Y   ='); MoveTo(col+26,GetY+5);
        OutText(' R        Red ');
      end;

      If ProgramName = 'Ltest' Then Begin
        MoveTo(col+25,GetY+30);
        OutText('e  = Y - E(Y) ='); MoveTo(col+34,GetY+5);
        OutText('R');
      end;

      If eR.rows = 3 then
GMatWrite(eR,4,1,'',X1+col+147,Y1+GetY-25,ErrReducedColor,7,0)
                  else
GMatWrite(eR,4,1,'',X1+col+147,Y1+GetY-20,ErrReducedColor,7,0);
      MoveTo(col,GetY+125);
      OutText('SSE  =  e    =  ');
      Str(SSER:4:2,msg); OutText(msg);
      MoveTo(col+1,GetY+5); OutText('   R     R');
      SetLineStyle(0,0,1);
      Line(col+60,GetY-9, col+60,GetY+2);
      Line(col+83,GetY-9, col+83,GetY+2);
      OutTextXY(col+87,GetY-8,'2');
```

```
      MoveTo(col,GetY+15);
      OutText('SSR = SSE  - SSE'); MoveTo(col+1,GetY+5);
      OutText('         R      F');

      MoveTo(col,GetY+16);
      If ProgramName = 'Bivariate' Then begin
        OutText('    =  Y - Y      -  e    '); MoveTo(col,GetY-5);
        OutText('           ^     2       2'); MoveTo(col+1,GetY+10);
        OutText('            Red       F  ');
      end;

      If ProgramName = 'Ltest' Then Begin
        OutText('    =  Y - E(Y)   -  e    '); MoveTo(col,GetY-5);
        OutText('       ^         2       2'); MoveTo(col+1,GetY+10);
        OutText('                        F  ');
      end;

      Line(col+52,GetY-11, col+52,GetY);
      Line(col+125,GetY-11, col+125,GetY);

      Line(col+164,GetY-9, col+164,GetY+2);
      Line(col+188,GetY-9, col+188,GetY+2);

      MoveTo(col,GetY+14);

      OutText('    =  e   -  e     =    ');
      Str(SSR:5:2,msg); OutText(msg);

      MoveTo(col+1,GetY+5);
      OutText('         R      F');
      SetLineStyle(0,0,1);
      Line(col+52,GetY-9, col+52,GetY+2);
      Line(col+75,GetY-9, col+75,GetY+2);

      Line(col+108,GetY-9, col+108,GetY+2);
      Line(col+131,GetY-9, col+131,GetY+2);
      OutTextXY(col+78,GetY-8,'2');
      OutTextXY(col+134,GetY-8,'2');


      MoveTo(col+25, GetY+20);
      OutText('df    = n - (p-1) = ');
      Str(dfR:2,msg); OutText(msg);
      MoveTo(col+1+25,GetY+5);
      OutText('  R');

      MoveTo(col+25, GetY+10);
      OutText('df    =   n - p   = ');
      Str(dfF:2,msg); OutText(msg);
      MoveTo(col+1+25,GetY+5);
      OutText('  F'); MoveTo(col+25,GetY+10);

      OutText('df    =  df  - df = ');
      Str(dfSSR:2,msg); OutText(msg);
      MoveTo(col+1+25,GetY+5);
      OutText('  SSR      R     F');

      col:=7; MoveTo(col,GetY+18);
      OutText('Fstar = (SSR/df   ) / (SSE  /df )'); MoveTo(col+1,
GetY+5);
      OutText('               SSR       F    F'); MoveTo(col, GetY+14);

      OutText('Fcrit = (1-'#224'; df     df )'); MoveTo(col+1, GetY+5);
      OutText('                SSR,   F'); col:=25;
```

```
      SetTextJustify(1,2);
      MoveTo((X2-X1) div 2,Y2-Y1-15); OutText('F3 Toggles this Screen
Off');
      SetColor(15);
      SetVisualPage(1);
      SetActivePage(0);
    end;
    RegressionScreenOn:=True;
  end
  else begin                              {If the Help Screen is }
    SetVisualPage(0);                     {Displayed then Erase it}
    SetActivePage(1);
    for i:= 1 to 2 do begin

      SetViewPort(0,0,GetMaxX,GetMaxY,True);
      PutImage(X1,Y1,RegressionPointer^,0);     {0=Copy}

      SetActivePage(0);
      SetVisualPage(1);
    end;

    RegressionScreenOn := False;
    SetColor(15);
    SetTextJustify(OldStyle.Horiz, OldStyle.Vert);

    Release(RegressionPointer);
  end;
end;  {procedure RegressionScreen}


Procedure MScreen;

const
  X1:Integer=360;
  Y1:Integer=14;              {Location of Window on Screen}
  X2:Integer=639;
  Y2:Integer=150;
  col:Integer=70;

var
  i: integer;
  Size: Word;
  ViewPort: ViewPortType;
  OldStyle: TextSettingsType;
  msg: string;

begin
  SetVisualPage(0);
  SetActivePage(1);

  GetTextSettings(OldStyle);
  SetTextJustify(1,1);

  If Not MScreenOn Then begin        {If the Help Screen is not}
    SetViewPort(0,0,GetMaxX,GetMaxY,True);      {Displayed then Display
it}
    Size := ImageSize(X1,Y1,X2,Y2);
    If Size>MemAvail then writeln(output,'Size=',Size,'
MemAvail=',MemAvail);
    Mark(MPointer);
    GetMem(MPointer, Size);
    GetImage(X1,Y1,X2,Y2,MPointer^);

    For i:= 1 to 2 do begin
```

```
      SetViewPort(X1,Y1,X2,Y2,True);
      GetViewSettings(ViewPort);
      SetFillStyle(1,15);
      Bar(0,0,X2-X1,Y2-Y1);              {Clear ViewPort}
      SetFillStyle(1,7);
      Bar(5,4,X2-X1-5,Y2-Y1-4);
      SetColor(0);
      SetTextJustify(1,1);
      OutTextXY((X2-X1) div 2,10,'M Projection Matrix');

      SetTextJustify(0,1);
      MoveTo(col,GetY+31);  SetColor(2);
      GMatWrite(M,4,1,'M.Full',X1+col,Y1+GetY,15,7,0);
MoveTo(col,GetY+85);
      GMatWrite(Mred,4,1,'M.Red ',X1+col,Y1+GetY,15,7,0);

      SetVisualPage(1);
      SetActivePage(0);
    end;
    MScreenOn:=True;
  end
  else begin                              {If the Help Screen is }
    for i:= 1 to 2 do begin

      SetViewPort(0,0,GetMaxX,GetMaxY,True);
      PutImage(X1,Y1,MPointer^,0);     {0=Copy}

      SetActivePage(0);
      SetVisualPage(1);
    end;

    MScreenOn := False;
    SetColor(15);
    SetTextJustify(OldStyle.Horiz, OldStyle.Vert);

    Release(MPointer);
  end;
end;  {procedure MScreen}

procedure chart(x1,yhi: Integer;       {Upper Left Location}
                x2,y2: Integer;        {Lower Right Location}
              Percent: Real;           {between 0 and 1}
                Color: Word;           {Color of chart}
                Title: String);        {Title of chart}

const
  AxisColor=15;     {White}
  MaxSize=0.80;     {At MaxSize, the bar will go up to this proportion }
                    {of the length of the Axis}

var
  OldStyle: TextSettingsType;
  OldView: ViewPortType;
  OldFill: FillSettingsType;
  msg: string;
  XWidth, Yheight, y1: Integer;
  i: Integer;
  UpperTitle,LowerTitle: String;
  WritePercent,Upper: Boolean;

begin
  If abs(Percent) > 1 then writeln(output,'Graph Percent should ',
    'be between 0 and 1.  Percent = ',Percent);
  If Percent > 0 then WritePercent := True
```

```
                else begin WritePercent := False; Percent :=
abs(Percent); end;

  GetTextSettings(OldStyle);
  GetViewSettings(OldView);
  GetFillSettings(OldFill);

  SetFillStyle(1, GetBkColor);
  Bar(x1,yhi, x2,y2);                    {Erase the old Graph}

  y1:=yhi+10;
  SetColor(AxisColor);
  SetLineStyle(0,0,1);

  Line(x1,y1,x1,y2);                  {Draw Y Axis}
  Line(x1,y2,x2,y2);                  {Draw X Axis}

  SetTextJustify(1,2);  {Center Horizontally & to the top Vertically}

  {The routine below looks for a decimal point and assumes that
characters
   following the decimal point will be placed as a subscript.
   Only 1 level of subscripting is supported.}

  Upper:=True;   {Title starts in normal Text}
  UpperTitle:=''; LowerTitle:='';

  for i:=1 to Length(Title) do
  If Copy(Title,i,1) = '.' then Upper:=Not Upper
    else    Case Upper of
                True  : begin UpperTitle:=UpperTitle+Copy(Title,i,1);
                              LowerTitle:=LowerTitle+' '; end;
                False : begin LowerTitle:=LowerTitle+Copy(Title,i,1);
                              UpperTitle:=UpperTitle+' '; end;
  end;


   {  writeln(output,'.',Title,'.');
      writeln(output,'.',UpperTitle,'.');
      writeln(output,'.',LowerTitle,'.');
      writeln(output,'  ');       }

  OutTextXY(x1+(x2-x1) div 2, Yhi, UpperTitle);
  OutTextXY(x1+(x2-x1) div 2+1, Yhi+4, LowerTitle);

  XWidth:=Round((x2-x1)*MaxSize*Percent);
  YHeight:=Round((y2-y1)*MaxSize*Percent);

  SetFillStyle(1,Color);              {Color of Bar Graph}
  Bar(x1+1,y2-1,x1+XWidth,            {Graph the chart}
      y2-YHeight);

  Str(Percent:4:2,msg);
  SetTextJustify(1,1);  {Center Vertically & Horizontally}
  If WritePercent then begin
{ If Percent>0.75 then begin
{   SetFillStyle(1,GetBkColor);                                    {Erase
a portion so that we can}
{   Bar(x1+XWidth div 2 - Length(msg)*4, y2-YHeight div 2 - 6,   {Write
out the percentage}
{       x1+XWidth div 2 + Length(msg)*4+1, y2-YHeight div 2 + 4);
    OutTextXY(x1+XWidth div 2, y2-YHeight div 2, msg); end
   else
```

```
    OutTextXY(Round(x1+Round((x2-x1)*0.65)), Round(y2-(y2-y1)*0.65),
msg);}

   SetFillStyle(1,BackgroundKolor);
{Erase a portion so that we can}
   Bar(x1+(x2-x1) div 2 - Length(msg)*4, y1 + (y2-y1) div 2 - 6,
{Write out the percentage}
       x1+(x2-x1) div 2 + Length(msg)*4+1, y1 + (y2-y1) div 2 + 4);
    OutTextXY(x1+(x2-x1) div 2, y1 + (y2-y1) div 2, msg); end;



 { setcolor(15);
  setLineStyle(0,0,1);
  rectangle(x1,yhi,x2,y2); }


  SetTextJustify(OldStyle.Horiz, OldStyle.Vert);
  SetFillStyle(OldFill.Pattern, OldFill.Color);
  With OldView do SetViewPort(x1,y1,x2,y2,Clip);

end;

function TimeInSeconds: Real;
var
  hr,min,sec,sec100: Word;

begin
  GetTime(hr,min,sec,sec100);
  TimeInSeconds:=hr*3600+min*60+sec+sec100/100;
end;

procedure BarGraphs;

const
  RowHeight=80;     {Height of graphs}
  X_Frame=0.05;     {Distance from Quadrant to dotted box}
  Y_Frame=0.05;
  X2_Frame=0.25;    {Keeps the quadrants seperate}
  Y2_Frame=0.25;

Type
  Location = Record X,Y: Integer; end;

var
  row3,Xoffset: Integer;
  UpperLeft,UpperRight,LowerLeft,LowerRight: Location;
  OldStyle: TextSettingsType;
  SSRS,dfSSRS,SSEFS,dfFS,msg: String;

begin
  UpperLeft.X:=Round((1+X_Frame)*Xhi*GetMaxX+2);
UpperLeft.Y:=Round((1+Y_Frame)*(1-Yhi)*GetMaxY-1);
  UpperRight.X:=Round((1-X_Frame)*GetMaxX); UpperRight.Y:=UpperLeft.Y;
  LowerLeft.X:=UpperLeft.X; LowerLeft.Y:=Round((1-Y_Frame)*GetMaxY);
  LowerRight.X:=Round((1-X_Frame)*GetMaxX);
LowerRight.Y:=Round((1-Y_Frame)*GetMaxY);

 {Writeln(output,'UpperLeft=',UpperLeft.X,', ',UpperLeft.Y);
  Writeln(output,'UpperRight=',UpperRight.X,', ',UpperRight.Y);
  Writeln(output,'LowerLeft=',LowerLeft.X,', ',LowerLeft.Y);
  Writeln(output,'LowerRight=',LowerRight.X,', ',LowerRight.Y);
  Writeln(output,'  ');   }
```

```
  {1,1}

chart(UpperLeft.X,UpperLeft.Y,Round((1-X2_Frame)*(UpperRight.X-UpperLeft
.X)) div 3 + UpperLeft.X,
        Round((1-Y2_Frame)*(LowerLeft.Y-UpperLeft.Y)) div 3 +
UpperLeft.Y, SSR/SSER, YhatColor, 'SSR');

  {2,1}
  chart(UpperLeft.X,UpperLeft.Y + (LowerLeft.Y-UpperLeft.Y) div 3,
        Round((1-X2_Frame)*(UpperRight.X-UpperLeft.X)) div 3 +
UpperLeft.X,
        LowerLeft.Y-Round((1+Y2_Frame)*(LowerLeft.Y-UpperLeft.Y)) div 3,
SSEF/SSER, ErrorColor, 'SSE.F');

  {3,1}
  chart(UpperLeft.X, LowerLeft.Y-(LowerLeft.Y-UpperLeft.Y) div 3,
        Round((1-X2_Frame)*(UpperRight.X-UpperLeft.X)) div 3 +
UpperLeft.X,
        LowerLeft.Y-Round(Y2_Frame*(LowerLeft.Y-UpperLeft.Y)) div 3,
SSTO/SSER, SSTOColor, 'SSTO');


  {1,2}
  chart(UpperLeft.X + (UpperRight.X-UpperLeft.X) div 3, UpperLeft.Y,
        UpperRight.X - Round((1+X2_Frame)*(UpperRight.X-UpperLeft.X))
div 3,
        Round((1-Y2_Frame)*(LowerLeft.Y-UpperLeft.Y)) div 3 +
UpperLeft.Y, -(SSR/SSER)/dfSSR, YhatColor, 'SSR/df.Num');

  {2,2}
  chart(UpperLeft.X + (UpperRight.X-UpperLeft.X) div 3,
        UpperLeft.Y + (LowerLeft.Y-UpperLeft.Y) div 3,
        UpperRight.X - Round((1+X2_Frame)*(UpperRight.X-UpperLeft.X))
div 3,
        LowerLeft.Y-Round((1+Y2_Frame)*(LowerLeft.Y-UpperLeft.Y)) div 3,
-(SSEF/SSER)/dfF, ErrorColor, 'SSE.F./df.Den');

  GetTextSettings(OldStyle);
  SetTextJustify(0,1);

  Str(SSR:4:1,SSRS);
  Str(dfSSR:1,dfSSRS);
  Str(SSEF:4:1,SSEFS);
  Str(dfF:1,dfFS);


  OutTextXY(UpperRight.X - (UpperRight.X-UpperLeft.X) div 3,
            UpperLeft.Y + (Round((1-Y2_Frame)*(LowerLeft.Y-UpperLeft.Y))
div 3) div 3,
            'SSR   =');

  OutTextXY(UpperRight.X - (UpperRight.X-UpperLeft.X) div 3 + 64,
            UpperLeft.Y + (Round((1-Y2_Frame)*(LowerLeft.Y-UpperLeft.Y))
div 3) div 3,
            SSRS);

  OutTextXY(UpperRight.X - (UpperRight.X-UpperLeft.X) div 3,
            UpperLeft.Y +
2*(Round((1-Y2_Frame)*(LowerLeft.Y-UpperLeft.Y)) div 3) div 3,
            'df    =');

  OutTextXY(UpperRight.X - (UpperRight.X-UpperLeft.X) div 3 + 80,
            UpperLeft.Y +
2*(Round((1-Y2_Frame)*(LowerLeft.Y-UpperLeft.Y)) div 3) div 3,
```

```
            dfSSRS);

  OutTextXY(UpperRight.X - (UpperRight.X-UpperLeft.X) div 3 + 1,
            UpperLeft.Y +
2*(Round((1-Y2_Frame)*(LowerLeft.Y-UpperLeft.Y)) div 3) div 3 + 4,
            '  Num');


  OutTextXY(UpperRight.X - (UpperRight.X-UpperLeft.X) div 3,
            UpperLeft.Y + (LowerLeft.Y-UpperLeft.Y) div 3 +
(Round((1-Y2_Frame)*(LowerLeft.Y-UpperLeft.Y)) div 3) div 3,
            'SSE   =');

  OutTextXY(UpperRight.X - (UpperRight.X-UpperLeft.X) div 3 + 1,
            UpperLeft.Y + (LowerLeft.Y-UpperLeft.Y) div 3 +
(Round((1-Y2_Frame)*(LowerLeft.Y-UpperLeft.Y)) div 3) div 3 + 4,
            '   F   ');

  OutTextXY(UpperRight.X - (UpperRight.X-UpperLeft.X) div 3 + 64,
            UpperLeft.Y + (LowerLeft.Y-UpperLeft.Y) div 3 +
(Round((1-Y2_Frame)*(LowerLeft.Y-UpperLeft.Y)) div 3) div 3,
            SSEFS);

  OutTextXY(UpperRight.X - (UpperRight.X-UpperLeft.X) div 3,
            UpperLeft.Y + (LowerLeft.Y-UpperLeft.Y) div 3 +
2*(Round((1-Y2_Frame)*(LowerLeft.Y-UpperLeft.Y)) div 3) div 3,
            'df    =');

  OutTextXY(UpperRight.X - (UpperRight.X-UpperLeft.X) div 3 + 1,
            UpperLeft.Y + (LowerLeft.Y-UpperLeft.Y) div 3 +
2*(Round((1-Y2_Frame)*(LowerLeft.Y-UpperLeft.Y)) div 3) div 3 + 4,
            '  Den');

  OutTextXY(UpperRight.X - (UpperRight.X-UpperLeft.X) div 3 + 80,
            UpperLeft.Y + (LowerLeft.Y-UpperLeft.Y) div 3 +
2*(Round((1-Y2_Frame)*(LowerLeft.Y-UpperLeft.Y)) div 3) div 3,
            dfFS);

  Xoffset:=25;

    If Fstar<>999 then Str(Fstar:4:2,msg)
    else msg:=#236;
    msg:='Fstar = '+msg;
    OutTextXY(UpperLeft.X + (UpperRight.X-UpperLeft.X) div 3 + Xoffset,
              LowerLeft.Y - (LowerLeft.Y-UpperLeft.Y) div 3+5, msg);

    Str(Pvalue:4:2,msg);
    msg:='P value = '+msg;
    OutTextXY(UpperLeft.X + (UpperRight.X-UpperLeft.X) div 3 + Xoffset,
              LowerLeft.Y - (LowerLeft.Y-UpperLeft.Y) div 3+17, msg);

    Str(Fcrit:4:2,msg);
    msg:='Fcrit = '+msg;
    OutTextXY(UpperLeft.X + (UpperRight.X-UpperLeft.X) div 3 + Xoffset,
              LowerLeft.Y - (LowerLeft.Y-UpperLeft.Y) div 3+17+17, msg);

    Str(Alpha.data[1,1]:4:2,msg);
    msg:='for '#224' = '+msg;
    OutTextXY(UpperLeft.X + (UpperRight.X-UpperLeft.X) div 3 + Xoffset,
              LowerLeft.Y - (LowerLeft.Y-UpperLeft.Y) div 3+17+17+12,
msg);

  SetTextJustify(OldStyle.Horiz, OldStyle.Vert);
```

```
end; {Procedure BarGraphs}

Procedure Screen(ScreenNumber: Integer);

Procedure ScreenCall(ScreenChar:String);

begin
  If ScreenChar = '1' Then Help;
  If ScreenChar = '2' Then ToggleKeyScreen;
  If ScreenChar = '3' Then RegressionScreen;
  If ScreenChar = '4' Then MScreen;
end; {Procedure ScreenCall}

var
  ScreenOn: Boolean;
  ScreenChar: String;
  i, Position, ScreenStackLength: Integer;

begin
  Str(ScreenNumber:1,ScreenChar);
  Position:=Pos(ScreenChar,ScreenStack);
  ScreenStackLength:=Length(ScreenStack);
  If ScreenChar = '0' Then Begin    {ESCape was pressed}
      ScreenCall(Copy(ScreenStack,Length(ScreenStack),1));
      ScreenStack:=Copy(ScreenStack,1,Length(ScreenStack)-1); end
  else
  If Position > 0 Then              {Screen is being displayed}
    While Pos(ScreenChar,ScreenStack) > 0 do begin
      ScreenCall(Copy(ScreenStack,Length(ScreenStack),1));
      ScreenStack:=Copy(ScreenStack,1,Length(ScreenStack)-1);
    end
  else Begin                         {Screen is not being Displayed}
    ScreenCall(ScreenChar);
    ScreenStack := ScreenStack + ScreenChar;
  end;
end;


procedure F9Procedure;

var
  i: Integer;

begin
  Writeln(Output,'Line #     From     To       Color');
  For I:=FirstLine to LastLine do
    With WorldLine[I] do
    Writeln(Output, '  ',I:2,'     ',Endpoint1:5,'    ',Endpoint2:5,'
 ',LineColor);
  Writeln(Output,' ');
  Writeln(Output,' ');
  Writeln(Output,'Point #    X        Y         Z');
  For I:=FirstPoint to LastPoint do
  With WorldEndpoint[I].Transform3DEndpointVar do
    Writeln(Output, '  ',I:2,'     ',WorldX:6,'    ',WorldZ:6,'
',WorldY:6);
end;


procedure KeyAction;

begin
Repeat
```

```
  MolePlot(BackgroundKolor);

(*-------------------------------------------------------------------------
---*)
  (* Get the keyboard status, check the status of the CTRL and BREAK (or
   *)
  (* CTRL and C) keys, and clear the keyboard buffer.
   *)

(*-------------------------------------------------------------------------
---*)
  AcroMole.GetKeyboardStatus(GetKeyboardStatusVar);
  AcroMole.CheckBreak(CheckBreakVar);
  AcroMole.ClearKeyboardBuffer(ClearKeyboardBufferVar);

  If (GetKeyboardStatusVar.Down[4] And 16384+1024)<>0        {Plus/Minus
key held down}
  Then MolePlusMinus;

  If (GetKeyboardStatusVar.Down[4] And (2048+8192))<>0 Then  {Left/Right
Arrows}
  MoleLeftRight;

  If (GetKeyboardStatusVar.Down[4] And 256<>0) or            {Up/Down
Arrows}
     (GetKeyboardStatusVar.Down[5] And 1 <> 0) Then
  MoleUpDown;

  If (GetKeyboardStatusVar.Down[4] And 128<>0) Then begin    {Home Key}
    Radius:=0.62;             {Initial Position}
    LayerScreen:=65535;       {Bring Everything Back}
    RotMat[0,0]:= 24945;
    RotMat[0,1]:= -6782;
    RotMat[0,2]:=-20124;
    RotMat[1,0]:=   712;
    RotMat[1,1]:= 31294;      {Rotation Matrix, Initial Camera Position}
    RotMat[1,2]:= -9664;
    RotMat[2,0]:= 21224;
    RotMat[2,1]:=  6921;
    RotMat[2,2]:= 23976;
  end;

  If (GetKeyboardStatusVar.Down[0] And 2<>0) and (Length(ScreenStack) >
0)   {ESCape Pressed}
     Then Begin Screen(0); GetKeyboardStatusVar.Down[0]:=0;
                GetKeyboardStatusVar.Missed[0]:=0; end;

  If (GetKeyboardStatusVar.Down[3] And 2048<>0) Then Screen(1);
{F1 Pressed, Help}

  If (GetKeyboardStatusVar.Down[3] And 4096<>0) Then Screen(2);
{F2 Pressed, ToggleKeyScreen}

  If GetKeyboardStatusVar.Down[4] And 2<>0 Then GetYFlag:=True;
{F7 Pressed}

  If GetKeyboardStatusVar.Down[4] And 4<>0 Then GetXFlag:=True;
{F8 Pressed}

{ If (GetKeyboardStatusVar.Down[4] And 8<>0) Then F9Procedure;
{F9 Pressed}

{ If GetKeyboardStatusVar.Down[4] And 16<>0 Then ListColors;        {F10
Pressed}
```

```
{Print out Current Colors}

  If (GetKeyboardStatusVar.Down[3] And 256<>0) and
     (GetKeyboardStatusVar.Down[1] And 16384<>0) Then              {Alt A
Pressed}
     If TimeInSeconds>(AxisFlipTime+ToggleDelay) then begin
      If LayerScreen And 2 <> 0 then LayerScreen:=LayerScreen And
(65535-2)
      else LayerScreen:=LayerScreen Or 2;
      AxisFliptime:=TimeInSeconds;
     end;

  If (GetKeyboardStatusVar.Down[3] And 256<>0) and
     (GetKeyboardStatusVar.Down[2] And 16384<>0) Then              {Alt C
Pressed}
     If TimeInSeconds>(CubeFlipTime+ToggleDelay) then begin
      If LayerScreen And 1 <> 0 then LayerScreen:=LayerScreen And
(65535-1)
      else LayerScreen:=LayerScreen Or 1;
      CubeFliptime:=TimeInSeconds;
     end;

  If (GetKeyboardStatusVar.Down[3] And 256<>0) and
     (GetKeyboardStatusVar.Down[2] And 8<>0) Then                  {Alt H
Pressed}
     If TimeInSeconds>(HFlipTime+ToggleDelay) then begin
      If LayerScreen And 256 <> 0 then LayerScreen:=LayerScreen And
(65535-256)
      else begin
            {ChangeColors(YhatRedColor); GenerateBivariatePoints;}
             LayerScreen:=LayerScreen Or 256; end;
      HFliptime:=TimeInSeconds;
     end;

  If (GetKeyboardStatusVar.Down[3] And 256<>0) and
     (GetKeyboardStatusVar.Down[2] And 8192<>0) Then               {Alt X
Pressed}
     If TimeInSeconds>(XFlipTime+ToggleDelay) then begin
      If LayerScreen And 8 <> 0 then LayerScreen:=LayerScreen And
(65535-8)
      else begin
            {ChangeColors(ModelSpaceColor); GenerateBivariatePoints;}
             LayerScreen:=LayerScreen Or 8; end;
      XFliptime:=TimeInSeconds;
     end;

  If (GetKeyboardStatusVar.Down[3] And 256<>0) and
     (GetKeyboardStatusVar.Down[1] And 32<>0) Then                 {Alt Y
Pressed}
     If TimeInSeconds>(YFlipTime+ToggleDelay) then begin
      If LayerScreen And 16 <> 0 then LayerScreen:=LayerScreen And
(65535-16)
      else begin
            {ChangeColors(ObservationColor); GenerateBivariatePoints;}
             LayerScreen:=LayerScreen Or 16; end;
      YFliptime:=TimeInSeconds;
     end;

  If (GetKeyboardStatusVar.Down[3] And 256<>0) and
     (GetKeyboardStatusVar.Down[0] And 4<>0) Then begin             {Alt
1 Pressed}
      SetVisualPage(0); readln; end;
```

```
  If ProgramName = 'Mean' Then                              {This version of
Alt-E applies}
  If (GetKeyboardStatusVar.Down[3] And 256<>0) and          {Only to the Mean
Program      }
     (GetKeyboardStatusVar.Down[1] And 4<>0) Then                    {Alt E
Pressed}
    If TimeInSeconds>(EFlipTime+ToggleDelay) then begin
      If LayerScreen And 32 <> 0 then LayerScreen:=LayerScreen And
(65535-32)
    else begin
          {ChangeColors(MuColor); GenerateDataPoints;}
           LayerScreen:=LayerScreen Or 32; end;
    EFliptime:=TimeInSeconds;
   end;

  If ProgramName = 'Ltest' Then Begin                       {This version of
Alt-E applies}
    If (GetKeyboardStatusVar.Down[3] And 256<>0) and        {Only to the
Ltest Program     }
       (GetKeyboardStatusVar.Down[1] And 4<>0) Then              {Alt E
Pressed}
      If TimeInSeconds>(EFlipTime+ToggleDelay) then begin
        If LayerScreen And 4096 <> 0 then LayerScreen:=LayerScreen And
(65535-4096)
      else begin
            {ChangeColors(MuColor); GenerateDataPoints;}
             LayerScreen:=LayerScreen Or 4096; end;
      EFliptime:=TimeInSeconds;
     end;
    If GetKeyboardStatusVar.Down[3] And 32768<>0 Then GetMuFlag:=True;
{F5 Pressed}
  end;

  If ProgramName = 'Bivariate' Then begin                   {These Routines
Apply only to }
                                                            {the Bivariate
Program        }

    If GetKeyboardStatusVar.Down[3] And 16384<>0 Then Screen(4);
{F4 Pressed, MScreen}

{Display Projection Matrix}
    If GetKeyboardStatusVar.Down[3] And 32768<>0 Then
GetBetaIFlag:=True; {F5 Pressed}

    If (GetKeyboardStatusVar.Down[3] And 256<>0) and
       (GetKeyboardStatusVar.Down[2] And 16<>0) Then            {Alt J
Pressed}
      If TimeInSeconds>(JFlipTime+ToggleDelay) then begin
        If LayerScreen And 512 <> 0 then LayerScreen:=LayerScreen And
(65535-512)
      else begin
            {ChangeColors(YhatFullColor); GenerateBivariatePoints;}
             LayerScreen:=LayerScreen Or 512; end;
      JFliptime:=TimeInSeconds;
     end;

    If (GetKeyboardStatusVar.Down[3] And 256<>0) and
       (GetKeyboardStatusVar.Down[1] And 64<>0) Then            {Alt U
Pressed}
      If TimeInSeconds>(UFlipTime+ToggleDelay) then begin
        If LayerScreen And 2048 <> 0 then LayerScreen:=LayerScreen And
(65535-2048)
      else begin
```

```
            {ChangeColors(Observation2Color); GenerateBivariatePoints;}
             LayerScreen:=LayerScreen Or 2048; end;
       UFliptime:=TimeInSeconds;
      end;


    If (GetKeyboardStatusVar.Down[3] And 256<>0) and
       (GetKeyboardStatusVar.Down[2] And 4096<>0) Then          {Alt Z
Pressed}
      If TimeInSeconds>(ZFlipTime+ToggleDelay) then begin
        If LayerScreen And 1024 <> 0 then LayerScreen:=LayerScreen And
(65535-1024)
      else begin
            {ChangeColors(ModelSpace2Color); GenerateBivariatePoints;}
             LayerScreen:=LayerScreen Or 1024; end;
      ZFliptime:=TimeInSeconds;
     end;
  end;

  If (ProgramName = 'Bivariate') or (ProgramName = 'Ltest') Then begin
{These Routines Apply only to  }
                                                          {the Ltest &
Bivariate Programs}

    If GetKeyboardStatusVar.Down[3] And 8192<>0 Then Screen(3);
{F3 Pressed, Regression Info}

{Display all Possible Colors}

    If GetKeyboardStatusVar.Down[4] And 1<>0 Then GetAlphaFlag:=True;
{F6 Pressed}

    If (GetKeyboardStatusVar.Down[3] And 256<>0) and
       (GetKeyboardStatusVar.Down[2] And 2<>0) Then          {Alt F
Pressed}
      If TimeInSeconds>(FFlipTime+ToggleDelay) then begin
        If LayerScreen And 32 <> 0 then LayerScreen:=LayerScreen And
(65535-32)
      else begin
            {ChangeColors(ErrorColor); GenerateBivariatePoints;}
             LayerScreen:=LayerScreen Or 32; end;
      FFliptime:=TimeInSeconds;
     end;

    If (GetKeyboardStatusVar.Down[3] And 256<>0) and
       (GetKeyboardStatusVar.Down[1] And 8<>0) Then          {Alt R
Pressed}
      If TimeInSeconds>(RFlipTime+ToggleDelay) then begin
        If LayerScreen And 64 <> 0 then LayerScreen:=LayerScreen And
(65535-64)
      else begin
            {ChangeColors(ErrReducedColor); GenerateBivariatePoints;}
             LayerScreen:=LayerScreen Or 64; end;
      RFliptime:=TimeInSeconds;
     end;
  end;

{  If (GetKeyboardStatusVar.Down[1] And 8192<>0) and           {Control
Key and}
{     (GetKeyboardStatusVar.Down[4] And (2048+8192)<>0) Then
{Left/Right Arrows}
{     MoleTranslateHorizontal;     }
```

```
(*-------------------------------------------------------------------
------*)
(* Keep looping till the user presses CTRL and BREAK, CTRL and C, or
ESC.     *)
(*-------------------------------------------------------------------
------*)
Until (CheckBreakVar.ReturnCode<>AMOkay) Or          (* CTRL BREAK or
CTRL C.   *)
      ((GetKeyboardStatusVar.Down[0] And 2)<>0) Or   (* ESC key held
down.      *)
      ((GetKeyboardStatusVar.Missed[0] And 2)<>0) Or (* ESC key tapped.
      *)
      GetXFlag Or GetYFlag Or                        (* X or Y matrix
needs updating*)
      GetBetaIFlag Or GetAlphaFlag Or GetMuFlag;

{Writeln(output,'CheckBreak=',CheckBreakVar.ReturnCode<>AMOkay,
                '  Escape Down=',(GetKeyboardStatusVar.Down[0] And 2)<>0,
                '  Escape Missed=',(GetKeyboardStatusVar.Missed[0] And
2)<>0);

Writeln(output,'XFlag=',GetXFlag,'  YFlag=',GetYFlag,'
BetaFlag=',GetBetaIFlag);}
end; {Procedure KeyAction}



end. {Unit Support}
```

```
Unit Support2;

Interface
uses TGlobals,Graph,Crt,MathMat,GraphMat;

function VectorLength(X,Y,Z:Real): Real;

Procedure GenerateArrowhead (ALine: Integer;     {Line Needing Arrow}
                         var X1,Y1,Z1,           {1st Arrow Segment}
                             X2,Y2,Z2:Integer); {2nd Arrow Segment}
Procedure GenerateAxis;

Procedure GenerateCube(X,Y,Z: Integer);

Procedure GenerateDataPoints;

Procedure GenerateBivariatePoints;


Implementation

function VectorLength;

begin VectorLength:=Sqrt(Sqr(X)+Sqr(Y)+Sqr(Z)); end;


Procedure GenerateCube(X,Y,Z: Integer);

var
  I,J,K,L:  Integer;      {Misc Dummy Variables}

Begin
L:=22;      {1st Point allocated to the Cube}
For I:=0 to 1 Do
  For J:= 0 to 1 Do
    For K:= 0 to 1 Do
      With WorldEndpoint[L] do begin
        Transform3DEndpointVar.WorldX:=(K*X*2)-X; {X,Y, & Z are the
relative}
        Transform3DEndpointVar.WorldY:=(J*Y*2)-Y; {Sizes of the object}
        Transform3DEndpointVar.WorldZ:=(I*Z*2)-Z;
{       Writeln(output,'X=',Transform3DEndpointVar.WorldX,'
Y=',Transform3DEndpointVar.WorldY,
        '  Z=',Transform3DEndpointVar.WorldZ); }

        Inc(L);
      End; {With}
(*-------------------------------------------------------------------------
------*)
(* Calculate which endpoints to connect to form the Cube's lines.
      *)
(*-------------------------------------------------------------------------
------*)
For L:=17 To 28 Do
  With WorldLine[L] Do begin
    Case L-17 Of
      0   : begin Endpoint1:=22+0; Endpoint2:=22+ 1; end;
      1   : begin Endpoint1:=22+1; Endpoint2:=22+ 3; end;
      2   : begin Endpoint1:=22+3; Endpoint2:=22+ 2; end;
      3   : begin Endpoint1:=22+2; Endpoint2:=22+ 0; end;
      4   : begin Endpoint1:=22+4; Endpoint2:=22+ 5; end;
      5   : begin Endpoint1:=22+5; Endpoint2:=22+ 7; end;
      6   : begin Endpoint1:=22+7; Endpoint2:=22+ 6; end;
      7   : begin Endpoint1:=22+6; Endpoint2:=22+ 4; end;
```

```
      8    : begin Endpoint1:=22+2; Endpoint2:=22+ 6; end;
      9    : begin Endpoint1:=22+0; Endpoint2:=22+ 4; end;
     10    : begin Endpoint1:=22+1; Endpoint2:=22+ 5; end;
     11    : begin Endpoint1:=22+3; Endpoint2:=22+ 7; end;
    end; {Case}
    LineColor:=LabelColor; Layer:=1; end; {With}

end; {Procedure GenerateCube}


Procedure GenerateArrowhead;

const
  base=0.10;      {base of arrow}
  awidth=10;      {half width of arrow}

var
  dx,dy,dz: real;  {direction vector}
  bx,by,bz: real;  {base coordinate along arrow line}
  vhead,vtail: integer;  {Endpoint Number of head and tail}
  vheadx,vheady,vheadz: real; {Head Coordinates}
  vtailx,vtaily,vtailz: real; {Tail Coordinates}
  CosAngle: real;       {Cosine of angle between vectors (In Radians)}
  orthox,orthoy,orthoz,olength: real; {Orthogonal Coordinates}
  dlength: real;  {length of vector}
  Point:Boolean;  {Not a vector but a point}
begin
  vhead:=WorldLine[ALine].Endpoint1;  {Vector Head Endpoint Number}
  vtail:=WorldLine[ALine].Endpoint2;  {Vector Tail Endpoint Number}

  vheadx:=WorldEndpoint[vhead].Transform3DEndpointVar.WorldX;
  vheady:=WorldEndpoint[vhead].Transform3DEndpointVar.WorldY;
{Coordinates of Head}
  vheadz:=WorldEndpoint[vhead].Transform3DEndpointVar.WorldZ;

  vtailx:=WorldEndpoint[vtail].Transform3DEndpointVar.WorldX;
  vtaily:=WorldEndpoint[vtail].Transform3DEndpointVar.WorldY;
{Coordinates of Tail}
  vtailz:=WorldEndpoint[vtail].Transform3DEndpointVar.WorldZ;

  If (vheadx=vtailx) and (vheady=vtaily) and (vheadz=vtailz) then
Point:=True
                                                          else
Point:=False;
  dx:=vheadx - vtailx;
  dy:=vheady - vtaily;     {Direction Vector}
  dz:=vheadz - vtailz;

  {Now we must see if either the head or tail is at the origin.
   If it is, we can shift if over.}
If not Point then begin
  If (vheadx=0) and (vheady=0) and (vheadz=0) then vheadx:=1;
  If (vtailx=0) and (vtaily=0) and (vtailz=0) then vtailx:=1;

  {Now let's see if the angle between the two vectors is very small.}

  Repeat
    CosAngle:=(vheadx*vtailx+vheady*vtaily+vheadz*vtailz)/

(VectorLength(vheadx,vheady,vheadz)*VectorLength(vtailx,vtaily,vtailz));

    If (abs(CosAngle) < 0.005) or (abs(CosAngle) > 0.995) then begin
       vtaily:=vtaily-1;          {This are just arbitrary used to}
       vtailz:=vtailz+1; end      {make the angle non parallel}
```

```
    else If (abs(CosAngle)>0.990) or (abs(CosAngle)<=0.010) then begin
      vtailx:=vtailx - dx;      {If the angle isn't large enough, widen
it.}
      vtaily:=vtaily - dy;      {But keep it in the same plane.}
      vtailz:=vtailz - dz; end;

  Until (abs(CosAngle) > 0.010) and (abs(CosAngle) <0.990);    {Angle
must be atleast 0.010 radians}
                              {For the Cross Product Routine}

  orthox:=vheady*vtailz-vheadz*vtaily;
  orthoy:=vheadz*vtailx-vheadx*vtailz;   {vector orthogonal to original}
  orthoz:=vheadx*vtaily-vheady*vtailx;   {two vectors}

  olength:=VectorLength(orthox,orthoy,orthoz);  {Orthogonal Vector
Length}
  dlength:=VectorLength(Dx,Dy,Dz);              {Vector Length}
  if dlength>0 then dlength:=Ln(dlength);  {'Nomalize' it}

  bx:=vheadx - base*dx;
  by:=vheady - base*dy;                    {'height' of triangle formed
by arrowhead}
  bz:=vheadz - base*dz;

  X1:=Round(bx+orthox*awidth*dlength/olength);
  Y1:=Round(by+orthoy*awidth*dlength/olength);    {Base of one of the
arrow lines}
  Z1:=Round(bz+orthoz*awidth*dlength/olength);

  X2:=Round(bx-orthox*awidth*dlength/olength);
  Y2:=Round(by-orthoy*awidth*dlength/olength);    {Base of one of the
arrow lines}
  Z2:=Round(bz-orthoz*awidth*dlength/olength);
  end
 else begin
  X1:=round(vheadx); X2:=X1;
  Y1:=round(vheady); Y2:=Y1;
  Z1:=round(vheadz); Z2:=Z1; end;
end; {Procedure GenerateArrowhead}


Procedure GenerateAxis;

Const

  Size:Real=0.05;  {Size of X,Y,Z relative to Axis}
  DistX:Real=1.13; {Distance from arrowhead to center of 'X'}
  DistY:Real=1.2; {Distance from arrowhead to center of 'Y'}
  DistZ:Real=1.09; {Distance from arrowhead to center of 'Z'}.
  Magnify:Real=1.7;   {Indicates relative length of Axes to object}

var
  I,J: Integer;          {Looping Variables}
  DX,DY,DZ: Integer;     {Dummy Variables}
  MX,MY,MZ: Real;
  Center: Real;          {Center of Label}
  Leg: Real;             {Label variable}

Begin
MX := Xmax; MY := Ymax; MZ := Zmax;
If Xmax<((MY+MZ)/3) then Xmax:=(MY+MZ)/3;
If Ymax<((MX+MZ)/3) then Ymax:=(MX+MZ)/3;
If Zmax<((MX+MY)/3) then Zmax:=(MX+MY)/3;
```

```
{Now generate the right sized cube}
GenerateCube(Round(Xmax),Round(Ymax),Round(Zmax));

Xmax:= Magnify*Xmax;
Ymax:= Magnify*Ymax;     {We want the Axes to go past the object}
Zmax:= Magnify*Zmax;

For I:= 0 to 16 do        {Note: If # of Axis Lines change, Update the
'16'}
  With WorldLine[I] do begin    {Endpoint1 = Head}
    Case I Of                   {Endpoint2 = Tail}
      0    : begin Endpoint1:=1; Endpoint2:=0;  end; {X-Axis}
      1    : begin Endpoint1:=2; Endpoint2:=0;  end; {Y-Axis}
      2    : begin Endpoint1:=3; Endpoint2:=0;  end; {Z-Axis}
      3    : begin Endpoint1:=1; Endpoint2:=4;  end; {X-Arrowhead}
      4    : begin Endpoint1:=1; Endpoint2:=5;  end; {X-Arrowhead}
      5    : begin Endpoint1:=2; Endpoint2:=6;  end; {Y-Arrowhead}
      6    : begin Endpoint1:=2; Endpoint2:=7;  end; {Y-Arrowhead}
      7    : begin Endpoint1:=3; Endpoint2:=8;  end; {Z-Arrowhead}
      8    : begin Endpoint1:=3; Endpoint2:=9;  end; {Z-Arrowhead}
      9    : begin Endpoint1:=12; Endpoint2:=13;  end; {'X'}
      10   : begin Endpoint1:=10; Endpoint2:=11;  end; {'X'}
      11   : begin Endpoint1:=16; Endpoint2:=14;  end; {'Z'}
      12   : begin Endpoint1:=14; Endpoint2:=15;  end; {'Z'}
      13   : begin Endpoint1:=15; Endpoint2:=17;  end; {'Z'}
      14   : begin Endpoint1:=18; Endpoint2:=19;  end; {'Y'}
      15   : begin Endpoint1:=19; Endpoint2:=20;  end; {'Y'}
      16   : begin Endpoint1:=19; Endpoint2:=21;  end; {'Y'}
    end;  {Case}
  LineColor:=15; Layer:=2; end;    {Color & Layer of Axis}

  For I:= 1 to 9 do
   With WorldEndpoint[I].Transform3DEndpointVar do
    Case I Of
      1    : begin WorldX:=Round(Xmax); WorldY:=0; WorldZ:=0; end;
{X-Axis}
      2    : begin WorldX:=0; WorldY:=0; WorldZ:=Round(Zmax); end;
{Y-Axis}
      3    : begin WorldX:=0; WorldY:=Round(Ymax); WorldZ:=0; end;
{Z-Axis}

             {X-Axis Arrowhead}
      4    : begin GenerateArrowHead(0,WorldX,WorldY,WorldZ,DX,DY,DZ);
end;
      5    : begin GenerateArrowHead(0,DX,DY,DZ,WorldX,WorldY,WorldZ);
end;

             {Y-Axis Arrowhead}
      6    : begin GenerateArrowHead(1,WorldX,WorldY,WorldZ,DX,DY,DZ);
end;
      7    : begin GenerateArrowHead(1,DX,DY,DZ,WorldX,WorldY,WorldZ);
end;

             {Z-Axis Arrowhead}
      8    : begin GenerateArrowHead(2,WorldX,WorldY,WorldZ,DX,DY,DZ);
end;
      9    : begin GenerateArrowHead(2,DX,DY,DZ,WorldX,WorldY,WorldZ);
end;

    end; {Case}

    Leg:=1.2*Size*Xmax; Center:=DistX*Xmax;
    For I:= 10 to 13 do
    With WorldEndpoint[I].Transform3DEndpointVar do begin
```

```
    Case I of
      10  : begin WorldX:=Round(Center+Leg); WorldY:=Round(-Leg);
WorldZ:=0; end;
      11  : begin WorldX:=Round(Center-Leg); WorldY:=Round(Leg);
WorldZ:=0; end;   {'X' Label}
      12  : begin WorldX:=Round(Center+Leg); WorldY:=Round(Leg);
WorldZ:=0; end;
      13  : begin WorldX:=Round(Center-Leg); WorldY:=Round(-Leg);
WorldZ:=0; end;
    end; {Case} end; {With}

    Leg:=1.2*Size*Zmax; Center:=DistY*Ymax;
    For I:= 14 to 17 do
    With WorldEndpoint[I].Transform3DEndpointVar do begin
    WorldZ:=0;
    Case I of
      14  : begin WorldX:=Round(+Leg); WorldY:=Round(Center+Leg); end;
      15  : begin WorldX:=Round(-Leg); WorldY:=Round(Center-Leg); end;
{'Y' Label}
      16  : begin WorldX:=Round(-Leg); WorldY:=Round(Center+Leg); end;
      17  : begin WorldX:=Round(+Leg); WorldY:=Round(Center-Leg); end;
    end; {Case} end; {With}

    Leg:=Size*Ymax; Center:=DistZ*Zmax;
    For I:= 18 to 21 do
    With WorldEndpoint[I].Transform3DEndpointVar do begin
    WorldX:=0;
    Case I of
      18  : begin WorldY:=Round(+Leg); WorldZ:=Round(Center+Leg); end;
      19  : begin WorldY:=0; WorldZ:=Round(Center); end;
{'Z' Label}
      20  : begin WorldY:=Round(+Leg); WorldZ:=Round(Center-Leg); end;
      21  : begin WorldY:=Round(-Leg); WorldZ:=Round(Center); end;
    end; {Case} end; {With}

end; {Procedure GenerateAxis}


Procedure GenerateDataPoints;

var
  I: Integer;   {Dummy Variable}
  Point,Endpoint1,Endpoint2: Integer;
  DX,DY,DZ: Integer;

Function GetMaxValue(i,j: Real): Real;

var
  k: real;

begin
  If i>=j then k:=i
          else k:=j;
  GetMaxValue:=k;
end; {Function GetMaxValue}



begin

{First, Generate the points for the actual vectors that will be plotted.
Note:  The first [0-16] lines are the Axes and labels
       The first [0-21] points are for the Axes and labels
```

```
        The next [17-28] lines are for the Cube
        The next [22-29] points are for the Cube

    Therefore, the actual data points start at Line FirstLine and point
    FirstPoint}

    With WorldEndpoint[0].Transform3DEndpointVar do begin
      WorldX:=0; WorldY:=0; WorldZ:=0; end;     {Origin}

    For I:= FirstLine to LastLine do
    With WorldLine[I] do                    {Endpoint1 = Head}
    Case I-FirstLine Of                     {Endpoint2 = Tail}
       0    : begin Layer:=8; Point:=FirstPoint; Endpoint1:=Point;
    Endpoint2:=0; LineColor:=ModelSpaceColor; end; {Model Space, X}
       1    : begin Layer:=8; Endpoint1:=Point; Endpoint2:=Point+1;
    LineColor:=ModelSpaceColor; end; {Arrowhead}
       2    : begin Layer:=8; Endpoint1:=Point; Endpoint2:=Point+2;
    LineColor:=ModelSpaceColor; end; {Arrowhead}

       3    : begin Layer:=16; Point:=FirstPoint+3; Endpoint1:=Point;
    Endpoint2:=0;
                   LineColor:=ObservationColor; end; {Observed Value, Y}
       4    : begin Layer:=16; Endpoint1:=Point; Endpoint2:=Point+1;
    LineColor:=ObservationColor; end; {Arrowhead}
       5    : begin Layer:=16; Endpoint1:=Point; Endpoint2:=Point+2;
    LineColor:=ObservationColor; end; {Arrowhead}

       6    : begin Layer:=32; Point:=FirstPoint+6; Endpoint1:=Point;
    Endpoint2:=0; LineColor:=ErrorColor; end; {Error Space,eF}
       7    : begin Layer:=32; Endpoint1:=Point; Endpoint2:=Point+1;
    LineColor:=ErrorColor; end; {Arrowhead}
       8    : begin Layer:=32; Endpoint1:=Point; Endpoint2:=Point+2;
    LineColor:=ErrorColor; end; {Arrowhead}

       9    : begin Layer:=256; Point:=FirstPoint+9; Endpoint1:=Point;
    Endpoint2:=0; LineColor:=YhatColor; end; {Yhat}
      10    : begin Layer:=256; Endpoint1:=Point; Endpoint2:=Point+1;
    LineColor:=YhatColor; end; {Arrowhead}
      11    : begin Layer:=256; Endpoint1:=Point; Endpoint2:=Point+2;
    LineColor:=YhatColor; end; {Arrowhead}

      12    : begin Layer:=32; Point:=FirstPoint+12; Endpoint1:=Point;
    Endpoint2:=FirstPoint+9; LineColor:=ErrorColor; end;
                           {Yhat + Error Space(eF)}
      13    : begin Layer:=32; Endpoint1:=Point; Endpoint2:=Point+1;
    LineColor:=ErrorColor; end; {Arrowhead}
      14    : begin Layer:=32; Endpoint1:=Point; Endpoint2:=Point+2;
    LineColor:=ErrorColor; end; {Arrowhead}

      15    : begin Layer:=4096; Point:=FirstPoint+15; Endpoint1:=Point;
    Endpoint2:=0; LineColor:=MuColor; end; {Mu,E(Y)}
      16    : begin Layer:=4096; Endpoint1:=Point; Endpoint2:=Point+1;
    LineColor:=MuColor; end; {Arrowhead}
      17    : begin Layer:=4096; Endpoint1:=Point; Endpoint2:=Point+2;
    LineColor:=MuColor; end; {Arrowhead}

      18    : begin Layer:=64; Point:=FirstPoint+15+3; Endpoint1:=Point;
    Endpoint2:=0;
                   LineColor:=eRColor; end; {eR}
      19    : begin Layer:=64; Endpoint1:=Point; Endpoint2:=Point+1;
    LineColor:=ERColor; end; {Arrowhead}
      20    : begin Layer:=64; Endpoint1:=Point; Endpoint2:=Point+2;
    LineColor:=ERColor; end; {Arrowhead}
```

```
  21    : begin Layer:=64; Point:=FirstPoint+15+6; Endpoint1:=Point;
Endpoint2:=45; LineColor:=ERColor; end; {eR + E(Y)}
  22    : begin Layer:=64; Endpoint1:=Point; Endpoint2:=Point+1;
LineColor:=ERColor; end; {Arrowhead}
  23    : begin Layer:=64; Endpoint1:=Point; Endpoint2:=Point+2;
LineColor:=ERColor; end; {Arrowhead}


end; {Case}

{Calculate Max Size}
Xmax:=0; Ymax:=0; Zmax:=0;
for i:=1 to X.cols do Xmax:=GetMaxValue(Xmax,Abs(X.data[1,i]));
for i:=1 to Y.cols do Xmax:=GetMaxValue(Xmax,Abs(Y.data[1,i]));
for i:=1 to Error.cols do Xmax:=GetMaxValue(Xmax,Abs(Error.data[1,i]));
for i:=1 to Yhat.cols do Xmax:=GetMaxValue(Xmax,Abs(Yhat.data[1,i]));
for i:=1 to Error2.cols do
Xmax:=GetMaxValue(Xmax,Abs(Error2.data[1,i]));

for i:=1 to X.cols do Ymax:=GetMaxValue(Ymax,Abs(X.data[2,i]));
for i:=1 to Y.cols do Ymax:=GetMaxValue(Ymax,Abs(Y.data[2,i]));
for i:=1 to Error.cols do Ymax:=GetMaxValue(Ymax,Abs(Error.data[2,i]));
for i:=1 to Yhat.cols do Ymax:=GetMaxValue(Ymax,Abs(Yhat.data[2,i]));
for i:=1 to Error2.cols do
Ymax:=GetMaxValue(Ymax,Abs(Error2.data[2,i]));

If X.Rows>2 then begin
for i:=1 to X.cols do Zmax:=GetMaxValue(Zmax,Abs(X.data[3,i]));
for i:=1 to Y.cols do Zmax:=GetMaxValue(Zmax,Abs(Y.data[3,i]));
for i:=1 to Error.cols do Zmax:=GetMaxValue(Zmax,Abs(Error.data[3,i]));
for i:=1 to Yhat.cols do Zmax:=GetMaxValue(Zmax,Abs(Yhat.data[3,i]));
for i:=1 to Error2.cols do
Zmax:=GetMaxValue(Zmax,Abs(Error2.data[3,i])); end;

If (Xmax>=Ymax) and (Xmax>=Zmax) Then Scale:=ScaleSize/Xmax;
If (Ymax>=Xmax) and (Ymax>=Zmax) Then Scale:=ScaleSize/Ymax;
If (Zmax>=Ymax) and (Zmax>=Xmax) Then Scale:=ScaleSize/Zmax;
{Writeln(output,'Scale=',scale,'  Xmax=',Xmax,'  Ymax=',Ymax,'
Zmax=',Zmax);
}
Xmax:=Xmax*Scale; Ymax:=Ymax*Scale; Zmax:=Zmax*Scale; {These track to
plotting maximums}

For I:= FirstPoint to LastPoint do
With WorldEndpoint[I].Transform3DEndpointVar do
Case I-FirstPoint Of
  0 : begin WorldX:=Round(Scale*X.data[1,1]);       {Locate Point for}
            WorldZ:=Round(Scale*X.data[2,1]);       {X, the Model Space}
            If X.rows=2 then WorldY:=0 else
              WorldY:=Round(Scale*X.data[3,1]); end;
  1 : begin GenerateArrowHead(29,WorldX,WorldY,WorldZ,DX,DY,DZ); end;
  2 : begin GenerateArrowHead(29,DX,DY,DZ,WorldX,WorldY,WorldZ); end;

  3 : begin WorldX:=Round(Scale*Y.data[1,1]);       {Locate Point for}
            WorldZ:=Round(Scale*Y.data[2,1]);       {Y, the observed
value}
            If Y.rows=2 then WorldY:=0 else
              WorldY:=Round(Scale*Y.data[3,1]); end;
  4 : begin GenerateArrowHead(32,WorldX,WorldY,WorldZ,DX,DY,DZ); end;
  5 : begin GenerateArrowHead(32,DX,DY,DZ,WorldX,WorldY,WorldZ); end;

  6 : begin WorldX:=Round(Scale*Error.data[1,1]);      {Locate Point
for}
            WorldZ:=Round(Scale*Error.data[2,1]);      {Error Space}
```

```
            If Error.rows=2 then WorldY:=0 else
              WorldY:=Round(Scale*Error.data[3,1]); end;
  7 : begin GenerateArrowHead(35,WorldX,WorldY,WorldZ,DX,DY,DZ); end;
  8 : begin GenerateArrowHead(35,DX,DY,DZ,WorldX,WorldY,WorldZ); end;

  9 : begin WorldX:=Round(Scale*Yhat.data[1,1]);      {Locate Point for}
            WorldZ:=Round(Scale*Yhat.data[2,1]);      {Yhat}
            If Yhat.rows=2 then WorldY:=0 else
              WorldY:=Round(Scale*Yhat.data[3,1]); end;
 10 : begin GenerateArrowHead(38,WorldX,WorldY,WorldZ,DX,DY,DZ); end;
 11 : begin GenerateArrowHead(38,DX,DY,DZ,WorldX,WorldY,WorldZ); end;

 12 : begin WorldX:=Round(Scale*Error2.data[1,1]);      {Locate Point
for}
            WorldZ:=Round(Scale*Error2.data[2,1]);      {Error2}
            If Error2.rows=2 then WorldY:=0 else
              WorldY:=Round(Scale*Error2.data[3,1]); end;
 13 : begin GenerateArrowHead(41,WorldX,WorldY,WorldZ,DX,DY,DZ); end;
 14 : begin GenerateArrowHead(41,DX,DY,DZ,WorldX,WorldY,WorldZ); end;

 15 : begin WorldX:=Round(Scale*Mu.data[1,1]);       {Locate Point for}
            WorldZ:=Round(Scale*Mu.data[2,1]);       {Mu, the Model
Space}
            If Mu.rows=2 then WorldY:=0 else
              WorldY:=Round(Scale*Mu.data[3,1]); end;
 16 : begin
GenerateArrowHead(FirstLine+15,WorldX,WorldY,WorldZ,DX,DY,DZ); end;
 17 : begin
GenerateArrowHead(FirstLine+15,DX,DY,DZ,WorldX,WorldY,WorldZ); end;

 18 : begin WorldX:=Round(Scale*eR.data[1,1]);       {Locate Point for}
            WorldZ:=Round(Scale*eR.data[2,1]);       {eR, the observed
value}
            If eR.rows=2 then WorldY:=0 else
              WorldY:=Round(Scale*eR.data[3,1]); end;
 19 : begin
GenerateArrowHead(FirstLine+15+3,WorldX,WorldY,WorldZ,DX,DY,DZ); end;
 20 : begin
GenerateArrowHead(FirstLine+15+3,DX,DY,DZ,WorldX,WorldY,WorldZ); end;

 21 : begin WorldX:=Round(Scale*eRplusY.data[1,1]);      {Locate Point
for}
            WorldZ:=Round(Scale*eRplusY.data[2,1]);      {eR + Yhat}
            If eRplusY.rows=2 then WorldY:=0 else
              WorldY:=Round(Scale*eRplusY.data[3,1]); end;
 22 : begin
GenerateArrowHead(FirstLine+15+6,WorldX,WorldY,WorldZ,DX,DY,DZ); end;
 23 : begin
GenerateArrowHead(FirstLine+15+6,DX,DY,DZ,WorldX,WorldY,WorldZ); end;

end; {Case/With/For}

GenerateAxis;
NumberOfEndpoints:=LastPoint+1;
NumberOfLines:=LastLine+1;
end; {Procedure GenerateDataPoints}


Procedure GenerateBivariatePoints;

var
  I: Integer;   {Dummy Variable}
  Point,Endpoint1,Endpoint2: Integer;
  DX,DY,DZ: Integer;
```

```
Function GetMaxValue(i,j: Real): Real;

var
  k: real;

begin
  If i>=j then k:=i
          else k:=j;
  GetMaxValue:=k;
end; {Function GetMaxValue}



begin

{First, Generate the points for the actual vectors that will be plotted.
Note:  The first [0-16] lines are the Axes and labels
       The first [0-21] points are for the Axes and labels

       The next [17-28] lines are for the Cube
       The next [22-29] points are for the Cube

Therefore, the actual data points start at Line FirstLine and point
FirstPoint}

With WorldEndpoint[0].Transform3DEndpointVar do begin
  WorldX:=0; WorldY:=0; WorldZ:=0; end;    {Origin}

For I:= FirstLine to LastLine do
With WorldLine[I] do                    {Endpoint1 = Head}
Case I-FirstLine Of                     {Endpoint2 = Tail}
   0   : begin Layer:=8; Point:=FirstPoint; Endpoint1:=Point;
Endpoint2:=0; LineColor:=ModelSpaceColor; end;
            {Model Space, X[1,i]}
   1   : begin Layer:=8; Endpoint1:=Point; Endpoint2:=Point+1;
LineColor:=ModelSpaceColor; end; {Arrowhead}
   2   : begin Layer:=8; Endpoint1:=Point; Endpoint2:=Point+2;
LineColor:=ModelSpaceColor; end; {Arrowhead}

   3   : begin Layer:=1024; Point:=FirstPoint+3; Endpoint1:=Point;
Endpoint2:=0;
              LineColor:=ModelSpaceColor; end; {X[2,i]}
   4   : begin Layer:=1024; Endpoint1:=Point; Endpoint2:=Point+1;
LineColor:=ModelSpaceColor; end; {Arrowhead}
   5   : begin Layer:=1024; Endpoint1:=Point; Endpoint2:=Point+2;
LineColor:=ModelSpaceColor; end; {Arrowhead}

   6   : begin Layer:=16; Point:=FirstPoint+6; Endpoint1:=Point;
Endpoint2:=0; LineColor:=ObservationColor; end; {Yfull}
   7   : begin Layer:=16; Endpoint1:=Point; Endpoint2:=Point+1;
LineColor:=ObservationColor; end; {Arrowhead}
   8   : begin Layer:=16; Endpoint1:=Point; Endpoint2:=Point+2;
LineColor:=ObservationColor; end; {Arrowhead}

   9   : begin Layer:=2048; Point:=FirstPoint+9; Endpoint1:=Point;
Endpoint2:=0; LineColor:=Observation2Color; end; {Yreduced}
  10   : begin Layer:=2048; Endpoint1:=Point; Endpoint2:=Point+1;
LineColor:=Observation2Color; end; {Arrowhead}
  11   : begin Layer:=2048; Endpoint1:=Point; Endpoint2:=Point+2;
LineColor:=Observation2Color; end; {Arrowhead}

  12   : begin Layer:=32; Point:=FirstPoint+12; Endpoint1:=Point;
Endpoint2:=0; LineColor:=ErrorColor; end;   {eF}
```

```
  13    : begin Layer:=32; Endpoint1:=Point; Endpoint2:=Point+1;
LineColor:=ErrorColor; end; {Arrowhead}
  14    : begin Layer:=32; Endpoint1:=Point; Endpoint2:=Point+2;
LineColor:=ErrorColor; end; {Arrowhead}

  15    : begin Layer:=64; Point:=FirstPoint+15; Endpoint1:=Point;
Endpoint2:=0; LineColor:=ErrReducedColor; end;   {eR}
  16    : begin Layer:=64; Endpoint1:=Point; Endpoint2:=Point+1;
LineColor:=ErrReducedColor; end; {Arrowhead}
  17    : begin Layer:=64; Endpoint1:=Point; Endpoint2:=Point+2;
LineColor:=ErrReducedColor; end; {Arrowhead}

  18    : begin Layer:=256; Point:=FirstPoint+18; Endpoint1:=Point;
Endpoint2:=0; LineColor:=YhatRedColor; end;   {YhatRed}
  19    : begin Layer:=256; Endpoint1:=Point; Endpoint2:=Point+1;
LineColor:=YhatRedColor; end; {Arrowhead}
  20    : begin Layer:=256; Endpoint1:=Point; Endpoint2:=Point+2;
LineColor:=YhatRedColor; end; {Arrowhead}

  21    : begin Layer:=512; Point:=FirstPoint+21; Endpoint1:=Point;
Endpoint2:=0; LineColor:=YhatFullColor; end;  {YhatFull}
  22    : begin Layer:=512; Endpoint1:=Point; Endpoint2:=Point+1;
LineColor:=YhatFullColor; end; {Arrowhead}
  23    : begin Layer:=512; Endpoint1:=Point; Endpoint2:=Point+2;
LineColor:=YhatFullColor; end; {Arrowhead}

  24    : begin Layer:=64; Point:=FirstPoint+24; Endpoint1:=Point;
Endpoint2:=48; LineColor:=ErrReducedColor; end;  {YhatRed+eR}
  25    : begin Layer:=64; Endpoint1:=Point; Endpoint2:=Point+1;
LineColor:=ErrReducedColor; end; {Arrowhead}
  26    : begin Layer:=64; Endpoint1:=Point; Endpoint2:=Point+2;
LineColor:=ErrReducedColor; end; {Arrowhead}

  27    : begin Layer:=32; Point:=FirstPoint+27; Endpoint1:=Point;
Endpoint2:=51; LineColor:=ErrorColor; end; {YhatFull+eF}
  28    : begin Layer:=32; Endpoint1:=Point; Endpoint2:=Point+1;
LineColor:=ErrorColor; end; {Arrowhead}
  29    : begin Layer:=32; Endpoint1:=Point; Endpoint2:=Point+2;
LineColor:=ErrorColor; end; {Arrowhead}

end; {Case}

{Calculate Max Size}
Xmax:=0; Ymax:=0; Zmax:=0;
for i:=1 to X.cols do Xmax:=GetMaxValue(Xmax,Abs(X.data[1,i]));
for i:=1 to Y.cols do Xmax:=GetMaxValue(Xmax,Abs(Y.data[1,i]));
for i:=1 to Yred.cols do Xmax:=GetMaxValue(Xmax,Abs(Yred.data[1,i]));
for i:=1 to eF.cols do Xmax:=GetMaxValue(Xmax,Abs(eF.data[1,i]));
for i:=1 to eR.cols do Xmax:=GetMaxValue(Xmax,Abs(eR.data[1,i]));

for i:=1 to X.cols do Ymax:=GetMaxValue(Ymax,Abs(X.data[2,i]));
for i:=1 to Y.cols do Ymax:=GetMaxValue(Ymax,Abs(Y.data[2,i]));
for i:=1 to Yred.cols do Ymax:=GetMaxValue(Ymax,Abs(Yred.data[2,i]));
for i:=1 to eF.cols do Ymax:=GetMaxValue(Ymax,Abs(eF.data[2,i]));
for i:=1 to eR.cols do Ymax:=GetMaxValue(Ymax,Abs(eR.data[2,i]));

If X.Rows>2 then begin
for i:=1 to X.cols do Zmax:=GetMaxValue(Zmax,Abs(X.data[3,i]));
for i:=1 to Y.cols do Zmax:=GetMaxValue(Zmax,Abs(Y.data[3,i]));
for i:=1 to Yred.cols do Zmax:=GetMaxValue(Zmax,Abs(Yred.data[3,i]));
for i:=1 to eF.cols do Zmax:=GetMaxValue(Zmax,Abs(eF.data[3,i]));
for i:=1 to eR.cols do Zmax:=GetMaxValue(Zmax,Abs(eR.data[3,i])); end;

If (Xmax>=Ymax) and (Xmax>=Zmax) Then Scale:=ScaleSize/Xmax;
```

```
If (Ymax>=Xmax) and (Ymax>=Zmax) Then Scale:=ScaleSize/Ymax;
If (Zmax>=Ymax) and (Zmax>=Xmax) Then Scale:=ScaleSize/Zmax;
{Writeln(output,'Scale=',scale,'  Xmax=',Xmax,'  Ymax=',Ymax,'
Zmax=',Zmax);
}
Xmax:=Xmax*Scale; Ymax:=Ymax*Scale; Zmax:=Zmax*Scale; {These track to
plotting maximums}

For I:= FirstPoint to LastPoint do
With WorldEndpoint[I].Transform3DEndpointVar do
Case I-FirstPoint Of
  0 : begin WorldX:=Round(Scale*X.data[1,1]);       {Locate Point for}
            WorldZ:=Round(Scale*X.data[2,1]);       {X[1,i], the Reduced
Space}
            WorldY:=Round(Scale*X.data[3,1]); end;
  1 : begin
GenerateArrowHead(FirstLine+0,WorldX,WorldY,WorldZ,DX,DY,DZ); end;
  2 : begin
GenerateArrowHead(FirstLine+0,DX,DY,DZ,WorldX,WorldY,WorldZ); end;

  3 : begin WorldX:=Round(Scale*X.data[1,2]);       {Locate Point for}
            WorldZ:=Round(Scale*X.data[2,2]);       {X[2,i], the 'Full'
Space}
            WorldY:=Round(Scale*X.data[3,2]); end;
  4 : begin
GenerateArrowHead(FirstLine+3,WorldX,WorldY,WorldZ,DX,DY,DZ); end;
  5 : begin
GenerateArrowHead(FirstLine+3,DX,DY,DZ,WorldX,WorldY,WorldZ); end;

  6 : begin WorldX:=Round(Scale*Y.data[1,1]);       {Locate Point for}
            WorldZ:=Round(Scale*Y.data[2,1]);       {Yfull}
            WorldY:=Round(Scale*Y.data[3,1]); end;
  7 : begin
GenerateArrowHead(FirstLine+6,WorldX,WorldY,WorldZ,DX,DY,DZ); end;
  8 : begin
GenerateArrowHead(FirstLine+6,DX,DY,DZ,WorldX,WorldY,WorldZ); end;

  9 : begin WorldX:=Round(Scale*Yred.data[1,1]);       {Locate Point
for}
            WorldZ:=Round(Scale*Yred.data[2,1]);       {Yreduced}
            WorldY:=Round(Scale*Yred.data[3,1]); end;
 10 : begin
GenerateArrowHead(FirstLine+9,WorldX,WorldY,WorldZ,DX,DY,DZ); end;
 11 : begin
GenerateArrowHead(FirstLine+9,DX,DY,DZ,WorldX,WorldY,WorldZ); end;

 12 : begin WorldX:=Round(Scale*eF.data[1,1]);      {Locate Point for}
            WorldZ:=Round(Scale*eF.data[2,1]);      {Error Space, Full}
            WorldY:=Round(Scale*eF.data[3,1]); end;
 13 : begin
GenerateArrowHead(FirstLine+12,WorldX,WorldY,WorldZ,DX,DY,DZ); end;
 14 : begin
GenerateArrowHead(FirstLine+12,DX,DY,DZ,WorldX,WorldY,WorldZ); end;

 15 : begin WorldX:=Round(Scale*eR.data[1,1]);      {Locate Point for}
            WorldZ:=Round(Scale*eR.data[2,1]);      {Error Space,
Reduced}
            WorldY:=Round(Scale*eR.data[3,1]); end;
 16 : begin
GenerateArrowHead(FirstLine+15,WorldX,WorldY,WorldZ,DX,DY,DZ); end;
 17 : begin
GenerateArrowHead(FirstLine+15,DX,DY,DZ,WorldX,WorldY,WorldZ); end;
```

```
 18 : begin WorldX:=Round(Scale*YhatRed.data[1,1]);      {Locate Point
for}
            WorldZ:=Round(Scale*YhatRed.data[2,1]);      {YhatReduced}
            WorldY:=Round(Scale*YhatRed.data[3,1]); end;
 19 : begin
GenerateArrowHead(FirstLine+18,WorldX,WorldY,WorldZ,DX,DY,DZ); end;
 20 : begin
GenerateArrowHead(FirstLine+18,DX,DY,DZ,WorldX,WorldY,WorldZ); end;

 21 : begin WorldX:=Round(Scale*YhatFull.data[1,1]);      {Locate Point
for}
            WorldZ:=Round(Scale*YhatFull.data[2,1]);      {YhatFull}
            WorldY:=Round(Scale*YhatFull.data[3,1]); end;
 22 : begin
GenerateArrowHead(FirstLine+21,WorldX,WorldY,WorldZ,DX,DY,DZ); end;
 23 : begin
GenerateArrowHead(FirstLine+21,DX,DY,DZ,WorldX,WorldY,WorldZ); end;

 24 : begin WorldX:=Round(Scale*YhatRedeR.data[1,1]);      {Locate Point
for}
            WorldZ:=Round(Scale*YhatRedeR.data[2,1]);      {YhatReduced
+ eR}
            WorldY:=Round(Scale*YhatRedeR.data[3,1]); end;
 25 : begin
GenerateArrowHead(FirstLine+24,WorldX,WorldY,WorldZ,DX,DY,DZ); end;
 26 : begin
GenerateArrowHead(FirstLine+24,DX,DY,DZ,WorldX,WorldY,WorldZ); end;

 27 : begin WorldX:=Round(Scale*YhatFulleF.data[1,1]);      {Locate
Point for}
            WorldZ:=Round(Scale*YhatFulleF.data[2,1]);      {YhatFull +
eF}
            WorldY:=Round(Scale*YhatFulleF.data[3,1]); end;
 28 : begin
GenerateArrowHead(FirstLine+27,WorldX,WorldY,WorldZ,DX,DY,DZ); end;
 29 : begin
GenerateArrowHead(FirstLine+27,DX,DY,DZ,WorldX,WorldY,WorldZ); end;

end; {Case/With/For}

GenerateAxis;
NumberOfEndpoints:=LastPoint+1;
NumberOfLines:=LastLine+1;
end; {Procedure GenerateBivariatePoints}

end.
```

```
unit Support3;

interface

uses Crt,Dos,MathMat,GraphMat,TGlobals,Support,Support2,Mole,Graph;

Procedure MeanRoutine;

Procedure LtestRoutine;

Procedure BivariateRoutine;

Procedure ExitRoutine;

Procedure LeaveProgram;

implementation

Procedure MeanRoutine;

var
  i,j:                Integer;
  SampNum:                  Char;
  msg: String;
  char_H,char_W:Integer;  {Heigth & Width of char in pixels}
  OldStyle: TextSettingsType;
  BetaX,row1,row2,row3,row4,row5: Integer;     {1st row on which the
data is displayed}
  SSE,SSR,SSTO,Sigma: Real;
  SSES,SSRS,SSTOS: String;
  SSEP,SSRP: Integer;
  ErrorT,SSEM,SSRM,YhatT:      matx;

{ Main Program }

{Sample Mean Section}
begin
  char_H:=TextHeight('H');
  char_W:=TextWidth('H');
  row1:=4*char_H;
  If FirstRunFlag Then Begin
    SetBkColor(BackgroundKolor);
    SetTextJustify(1,1);
    OutTextXY (GetMaxX div 2,char_H,'Ordinary Least Squares Regression
for the Sample Mean and Variance');

    repeat
      LastLine:=43; LastPoint:=44;     {These show which vectors will be
displayed}
      Msg:='Do you want the sample size to be 2 or 3?';
      OutTextXY (GetMaxX div 2,Round(5*1.2*char_H),Msg);
      SampNum:= Readkey;
      OutTextXY
(Round(1.05*(GetMaxX+TextWidth(Msg))/2),Round(5*1.2*char_H),SampNum);
      OutTextXY (GetMaxX div 2,Round(6*1.2*char_H),'
                                ');
      case SampNum of
       '2'    :  begin Y.rows:=2; Y.cols:=1; X.rows:=2; X.cols:=1; end;
       '3'    :  begin Y.rows:=3; Y.cols:=1; X.rows:=3; X.cols:=1; end;
       else      OutTextXY (GetMaxX div 2,Round(6*1.2*char_H),'Sorry,
only a sample size of 2 or 3 are demonstrated.');
      end;
    until (SampNum = '2') or (SampNum = '3');
```

```
    SetFillStyle(1,BackgroundKolor);   {Solid,BackgroundKolor}
    Bar(0,Round(4.1*char_H),GetMaxX,GetMaxY);       {Clear the Screen}
    Mat_Zero(Y); Mat_Zero(X);
    For i := 1 to X.rows do X.data[i,1] := 1;
    GetXFlag:=True; GetYFlag:=True;
    FirstRunFlag := False;
  end;
(***************** Input Routines *****************)
  If GetYFlag then
    Repeat

GMatInput(Y,4,1,'Y',15,row1,ObservationColor,BackgroundKolor,MatForegrou
nd);       {***** Get Y *****}
      GetYFlag:=False;
    Until (abs(Y.data[1,1]) + abs(Y.data[2,1]) + abs(Y.data[Y.rows,1]))
> 0;

  If GetXFlag then begin

GMatInput(X,4,1,'X',100,row1,ModelSpaceColor,BackgroundKolor,MatForegrou
nd);       {***** Get X *****}
    GetXFlag:=False; end;
(******************** Input Routines ******************)

  MathRoutines;

{Now that all the calculations are done, we must generate a list
of points and lines to send to AcroMole for plotting.}

   GenerateDataPoints;

   SetActivePage(1);
   SetViewPort(0,0,GetMaxX,GetMaxY,True);
   SetTextJustify(1,1);
   SetColor(15);  {White}
   OutTextXY (GetMaxX div 2,char_H,'Ordinary Least Squares Regression
for the Sample Mean and Variance');

GMatWrite(Y,4,1,'Y',15,row1,ObservationColor,BackgroundKolor,MatForegrou
nd);

GMatWrite(X,4,1,'X',100,row1,ModelSpaceColor,BackgroundKolor,MatForegrou
nd);
   For i:= 1 downto 0 do begin     {Write to Both Buffers}
     SetActivePage(i);
     GMatWrite(M,4,1,'M',205,row1,15,BackgroundKolor,MatForeground);
     If M.rows=3 then begin
       OutTextXY(282,row1-7,'X[X''X]  X''');
OutTextXY(297,row1-12,'-1'); end
     else begin
       OutTextXY(257,row1-7,'X[X''X]  X''');
OutTextXY(272,row1-12,'-1'); end;

     BetaX:=380;

GMatWrite(Betahat,4,1,#225,BetaX,row1,15,BackgroundKolor,MatForeground);
SetColor(15); OutTextXY(BetaX-4,row1+5,'^');
     SetColor(LabelColor); OutTextXY(BetaX+25,row1-7,'[X''X]  X''Y');
OutTextXY(BetaX+31,row1-12,'-1');

     row2:=11*char_H;
     OutTextXY(120,row2,'Dimensions');
     SetTextJustify(0,1);
```

```
     OutTextXY(1,round(row2+1.5*char_H),'Sample Space     =   n   = [
]');
     Str(X.rows,msg); OutTextXY(225,round(row2+1.5*char_H),msg);
     OutTextXY(1,round(row2+2.7*char_H),'Estimation Space =   p   = [
]');
     Str(X.cols,msg); OutTextXY(225,round(row2+2.7*char_H),msg);
     OutTextXY(1,round(row2+3.9*char_H),'Error Space      = n - p = [
]');
     Str(X.rows-X.cols,msg); OutTextXY(225,round(row2+3.9*char_H),msg);
     SetTextJustify(1,1);


GMatWrite(Yhat,4,1,'Y',300,row2,YhatColor,BackgroundKolor,MatForeground)
; SetColor(15);

     If Y.rows=3 then begin
       OutTextXY(297,row2+12,'^'); SetColor(LabelColor);
OutTextXY(337,row2-7,'MY=X'#225);
       OutTextXY(353,row2-12,'^'); end
     else begin
       OutTextXY(296,row2+8,'^'); SetColor(LabelColor);
OutTextXY(338,row2-8,'MY=X'#225);
       OutTextXY(354,row2-13,'^'); end;

     SetColor(LabelColor);
     OutTextXY(335,row2+48,'Projection of Y onto');
     OutTextXY(335,row2+56,'the Estimation Space');


GMatWrite(Error,4,1,'Error',510,row2,ErrorColor,BackgroundKolor,MatForeg
round);
     SetColor(LabelColor);
     OutTextXY(557,row2-7,'(I-M)Y=Y-Y'); OutTextXY(593,row2-12,'^');
     OutTextXY(540,row2+48,'Projection of Y onto');
     OutTextXY(540,row2+56,'the Error Space');

     Mat_Transpose(Error,ErrorT);
     MatMult(Error,ErrorT,SSEM);
     SSE:=Sqrt(SSEM.Data[1,1]);
     Mat_Transpose(Yhat,YhatT);
     MatMult(Yhat,YhatT,SSRM);
     SSR:=Sqrt(SSRM.Data[1,1]);
     SSTO:=SSR+SSE;
     SSEP:=Round(100*SSE/SSTO);     {Percent SSE}
     SSRP:=Round(100*SSR/SSTO);     {Percent SSR}
     Sigma:=SSE/(X.rows-X.cols);

     SetFillStyle(1,GetBkColor); Bar(584,row1+1,639,row1+11);  {Erase
old Variance}
     Str(Sigma:4:1,msg); If Sigma>99 Then Str(Sigma:4:0,msg);
     OutTextXY(605,row1+7,msg);
     OutTextXY(500,row1,'SSE');
     SetLineStyle(0,0,1);

     Line(487,row1+6,513,row1+6);
     OutTextXY(500,row1+11,'n-p');
     OutTextXY(550,row1+7,'= '#229' {Y} =');
     OutTextXY(542,row1+3,'2');
     OutTextXY(534,row1+3,'^');
     row3:=31*char_H;
     chart(510,row3-80,630,row3, SSEP/100, ErrorColor, 'SSE');
     chart(370,row3-80,500,row3, SSRP/100, YhatColor, 'SSR');

     row4:=35*char_h;
```

```
      OutTextXY(505,row4,'Y      Y     Error');
      OutTextXY(493,row4-5,'^');
      OutTextXY(518,row4-5,'2      2         2');
      OutTextXY(466,row4,'='); OutTextXY(520,row4,'+');


      Line(487,row4+3,487,row4-5);    {Yhat}        {*************}
      Line(500,row4+3,500,row4-5);

      Line(444,row4+3,444,row4-5);    {Y}           {Absolute Value
Signs}
      Line(431,row4+3,431,row4-5);

      Line(580,row4+3,580,row4-5);    {Error}       {*************}
      Line(535,row4+3,535,row4-5);

      Str(SSR:4:1,SSRS); Str(SSE:4:1,SSES); Str(SSTO:4:1,SSTOS);
      If SSR>99 Then Str(SSR:4:0,SSRS);
      If SSE>99 Then Str(SSR:4:0,SSES);
      If SSTO>99 Then Str(SSR:4:0,SSTOS);

      OutTextXY(495,row4+16,'SSTO   SSR     SSE');
      OutTextXY(466,row4+16,'='); OutTextXY(520,row4+16,'+');

      SetFillStyle(1,GetBkColor); Bar(415,row4+23,575,row4+33);  {Erase
old Stats}
      OutTextXY(438,row4+29,SSTOS);
      OutTextXY(485,row4+29,SSRS);      {Display values for these
statistics}
      OutTextXY(550,row4+29,SSES);

      SetFillStyle(1,LabelColor);
      Bar(430,335,580,345); SetColor(0);
      OutTextXY(505,340,'Press F1 for Help');
      SetColor(15);

      SetLineStyle(2,0,1);                        {Block around spin area}
      MoveTo(round(Xlo*GetMaxX),round((1-Yhi)*GetMaxY)-1);
      LineTo(round(Xhi*GetMaxX)+2,round((1-Yhi)*GetMaxY)-1);
      LineTo(round(Xhi*GetMaxX)+2,round((1-Ylo)*GetMaxY));

Line(round(Xhi*GetMaxX)+2,round((1-Yhi)*GetMaxY)-1,GetMaxX,round((1-Yhi)
*GetMaxY)-1);
   end;
end; {Procedure MeanRoutine}


Procedure LtestRoutine;

var
  i,j:                Integer;
  SampNum: Char;
  Output: Text;
  msg: String;
  char_H,char_W:Integer;  {Heigth & Width of char in pixels}
  OldStyle: TextSettingsType;
  BetaX,row1a,row1b,row1c,row2a,row2b,row3,row3h,row4,row5: Integer;
{1st row on which the data is displayed}
  SSE,SSR,Sigma: Real;
  SSES,SSRS,SSTOS: String;
  SSEP,SSRP: Integer;
  ErrorT,SSEM,SSRM,YhatT:      matx;

{ Main Program }
```

```
begin
  char_H:=TextHeight('H');
  char_W:=TextWidth('H');
  row1a:=round(8.5*char_H);
  row1b:=4*char_H;
  row1c:=4*char_H;
  SetTextJustify(1,1);
  OutTextXY (GetMaxX div 2,char_H,'The General Linear Test for the
Population Mean');

  If FirstRunFlag Then Begin
    repeat
      Msg:='Do you want the sample size to be 2 or 3?';
      OutTextXY (GetMaxX div 2,Round(5*1.2*char_H),Msg);
      SampNum:= Readkey;
      OutTextXY
(Round(1.05*(GetMaxX+TextWidth(Msg))/2),Round(5*1.2*char_H),SampNum);
      OutTextXY (GetMaxX div 2,Round(6*1.2*char_H),'
                                  ');
      Case SampNum of
          '2'    :  begin Y.rows:=2; Y.cols:=1; X.rows:=2; X.cols:=1;
Mu.rows:=2;
                        LastLine:=52; LastPoint:=54; end;
          '3'    :  begin Y.rows:=3; Y.cols:=1; X.rows:=3; X.cols:=1;
Mu.rows:=3;
                        LastLine:=52; LastPoint:=54; end;
          else      OutTextXY (GetMaxX div 2,Round(6*1.2*char_H),'Sorry,
only a sample size of 2 or 3 are demonstrated.');
      end;
      Alpha.rows:=1; Alpha.cols:=1; Mu.cols:=1; Mu0.rows:=1; Mu0:cols:=1
    until (SampNum = '2') or (SampNum = '3');

    SetFillStyle(1,BackgroundKolor);   {Solid,Blue Background}
    Bar(0,Round(4.1*char_H),GetMaxX,GetMaxY);       {Clear the Screen}
    Mat_Zero(Y); Mat_Zero(X); Mat_Zero(Mu); Mat_Zero(Mu0);
Alpha.data[1,1]:=0.05;
    For i := 1 to X.rows do X.data[i,1] := 1;
    GetXFlag:=True; GetYFlag:=True; GetMuFlag:=True; GetAlphaFlag:=True;

    OutTextXY(50, row1c-10,'Ho: C'#230' = '#230'o');
    FirstRunFlag:= False;
  end;

(***************** Input Routines *****************)
  If GetMuFlag then begin

GMatInput(Mu0,4,1,#230'o',15,row1c,MuColor,BackgroundKolor,MatForeground
);       {***** Get Mu0 *****}
    GetMuFlag:=False;
    For I:=1 to Mu.rows do Mu.data[i,1]:=Mu0.data[1,1]; end;

  If GetAlphaFlag then begin

GMatInput(Alpha,4,2,#224,100,row1c,AlphaColor,BackgroundKolor,MatForegro
und);       {***** Get Alpha *****}
    GetAlphaFlag:=False; end;

  If GetYFlag then
    Repeat

GMatInput(Y,4,1,'Y',15,row1a,ObservationColor,BackgroundKolor,MatForegro
und);       {***** Get Y *****}
      GetYFlag:=False;
```

```
    Until (abs(Y.data[1,1]) + abs(Y.data[2,1]) + abs(Y.data[Y.rows,1]))
> 0;

  If GetXFlag then begin

GMatInput(X,4,1,'X',100,row1a,ModelSpaceColor,BackgroundKolor,MatForegro
und);       {***** Get X *****}
    GetXFlag:=False; end;
(******************** End of Input Routines ******************)
  MathRoutines;
  RegressionMath;    {More Calculations}

{Now that all the calculations are done, we must generate a list
of points and lines to send to AcroMole for plotting.}

  GenerateDataPoints;

  SetActivePage(1);
  SetViewPort(0,0,GetMaxX,GetMaxY,True);
  SetTextJustify(1,1);
  SetColor(15);  {White}
  OutTextXY (GetMaxX div 2,char_H,'The General Linear Test for the
Population Mean');
  OutTextXY(50, row1c-10,'Ho: C'#230' = '#230'o');


GMatWrite(Mu0,4,1,#230'o',15,row1c,MuColor,BackgroundKolor,MatForeground
);

GMatWrite(Alpha,4,2,#224,100,row1c,AlphaColor,BackgroundKolor,MatForegro
und);

GMatWrite(Y,4,1,'Y',15,row1a,ObservationColor,BackgroundKolor,MatForegro
und);

GMatWrite(X,4,1,'X',100,row1a,ModelSpaceColor,BackgroundKolor,MatForegro
und);

  For i:= 1 downto 0 do begin     {Write to Both Buffers}
    SetActivePage(i);
    GMatWrite(M,4,1,'M',205,row1b,15,BackgroundKolor,MatForeground);
    If M.rows=3 then begin
      OutTextXY(282,row1b-7,'X[X''X]  X''');
OutTextXY(297,row1b-12,'-1'); end
    else begin
      OutTextXY(257,row1b-7,'X[X''X]  X''');
OutTextXY(272,row1b-12,'-1'); end;

    BetaX:=380;

GMatWrite(Betahat,4,1,#225,BetaX,row1b,15,BackgroundKolor,MatForeground)
; SetColor(15); OutTextXY(BetaX-4,row1b+5,'^');
    SetColor(LabelColor); OutTextXY(BetaX+25,row1b-7,'[X''X]  X''Y');
OutTextXY(BetaX+31,row1b-12,'-1');

    row2a:=round(14.5*char_H);
    OutTextXY(120,row2a,'Dimensions');
    SetTextJustify(0,1);
    OutTextXY(1,round(row2a+1.5*char_H),'Sample Space     =   n   = [
]');
    Str(X.rows,msg); OutTextXY(225,round(row2a+1.5*char_H),msg);
    OutTextXY(1,round(row2a+2.7*char_H),'Estimation Space =   p   = [
]');
    Str(X.cols,msg); OutTextXY(225,round(row2a+2.7*char_H),msg);
```

```
    OutTextXY(1,round(row2a+3.9*char_H),'Error Space      = n - p = [
]');
    Str(X.rows-X.cols,msg); OutTextXY(225,round(row2a+3.9*char_H),msg);
    SetTextJustify(1,1);

    row2b:=11*char_H;


GMatWrite(Mu,4,1,'E(Y)',225,row2b,MuColor,BackgroundKolor,MatForeground)
;


GMatWrite(eR,4,1,'e.R',325,row2b,eRColor,BackgroundKolor,MatForeground);
    SetColor(LabelColor);
    OutTextXY(367,row2b-7,'Y-E(Y)');


GMatWrite(Yhat,4,1,'Y',425,row2b,YhatColor,BackgroundKolor,MatForeground
);
    SetColor(LabelColor);
    OutTextXY(462,row2b-7,'MY=X'#225);
    OutTextXY(478,row2b-12,'^');
    SetColor(15);
    If Y.rows=3 then OutTextXY(422,row2b+12,'^')
                else OutTextXY(422,row2b+8,'^');


GMatWrite(eF,4,1,'e.F',525,row2b,ErrorColor,BackgroundKolor,MatForegroun
d);
    SetColor(LabelColor);
    OutTextXY(569,row2b-7,'Y-Y');
    OutTextXY(569,row2b-13,'  ^');

{*****  Bar Graphs  ******}
    row3:=31*char_H;
    row3h:=80;
    SetFillStyle(1,GetBkColor);
    Bar(round(Xlo*GetMaxX)+1,round((1-Yhi)*GetMaxY),GetMaxX,GetMaxY);
    BarGraphs;

    SetFillStyle(1,LabelColor);
    Bar(430,335,580,345); SetColor(0);
    OutTextXY(505,340,'Press F1 for Help');
    SetColor(15);

    SetLineStyle(2,0,1);                          {Block around spin area}
    MoveTo(round(Xlo*GetMaxX),round((1-Yhi)*GetMaxY)-1);
    LineTo(round(Xhi*GetMaxX)+2,round((1-Yhi)*GetMaxY)-1);
    LineTo(round(Xhi*GetMaxX)+2,round((1-Ylo)*GetMaxY));

Line(round(Xhi*GetMaxX)+2,round((1-Yhi)*GetMaxY)-1,GetMaxX,round((1-Yhi)
*GetMaxY)-1);
  end;
end; {Procdure LtestRoutine}

Procedure BivariateRoutine;

var
  i,j:                Integer;
  msg: String;
  char_H,char_W:Integer;  {Heigth & Width of char in pixels}
  BetaX,row1,row2,row2a,row2b,row3,row3h,row4,row5: Integer;
  SSE,SSR: Real;
  SSES,SSRS,SSTOS: String;
```

```
  SSEP,SSRP: Integer;
  ErrorT,SSEM,SSRM,YhatT:      matx;

begin
  char_H:=TextHeight('H');
  char_W:=TextWidth('H');
  row1:=round(4.5*char_H);
  row2:=row1 + 44;

  Y.rows:=3; Y.cols:=1;
  Yred.rows:=3; Yred.cols:=1;
  X.rows:=3; X.cols:=2;
  Xred.rows:=3; Xred.cols:=1;
  Beta1.rows:=1; Beta1.cols:=1;
  Alpha.rows:=1; Alpha.cols:=1;

  LastLine:=58; LastPoint:=59;

  If FirstRunFlag Then Begin
    Mat_Zero(Y); Mat_Zero(X); Mat_Zero(Yred); Mat_Zero(Xred);
Mat_Zero(Beta1); Alpha.data[1,1]:=0.05;
    For i := 1 to X.rows do begin X.data[i,1] := 1; Xred.data[i,1] := 1;
end;
    GetXFlag:=True; GetYFlag:=True; GetBeta1Flag:=True;
GetAlphaFlag:=True;
    SetTextJustify(1,1);
    OutTextXY (GetMaxX div 2,char_H,'Ordinary Linear Simple Regression
with One Predictor Variable');
    OutTextXY(12,row1+97,'Ho:');
    FirstRunFlag := False;
  end;

(***************** Input Routines *****************)
  If GetBeta1Flag then begin
    GMatInput(Beta1,4,1,#225'10
',44,row1+88,15,BackgroundKolor,MatForeground);       {***** Get Beta1
*****}
    GetBeta1Flag:=False; end;

  If GetAlphaFlag then begin

GMatInput(Alpha,4,2,#224,150,row1+88,15,BackgroundKolor,MatForeground);
    {***** Get Beta1 *****}
    GetAlphaFlag:=False; end;

  If GetYFlag then
    Repeat

GMatInput(Y,4,1,'Y.Full',33,row1,ObservationColor,BackgroundKolor,MatFor
eground);       {***** Get Y *****}
      GetYFlag:=False;
    Until (abs(Y.data[1,1]) + abs(Y.data[2,1]) + abs(Y.data[Y.rows,1]))
> 0;

  If GetXFlag then begin
    If X.cols = 1 Then

GMatInput(X,4,1,'X.Full',150,row1,ModelSpaceColor,BackgroundKolor,MatFor
eground)        {***** Get X *****}
    else
      Repeat

GMatInput(X,4,1,'X.Full',150,row1,ModelSpaceColor,BackgroundKolor,MatFor
eground)        {***** Get X *****}
```

```
      Until (abs(X.data[1,2]) + abs(X.data[2,2]) + abs(X.data[3,2])) >
0;
    GetXFlag:=False; end;

(******************** End of Input Routines ******************)
  BivariateMath;    {Calculations}

{Now that all the calculations are done, we must generate a list
of points and lines to send to AcroMole for plotting.}

  GenerateBivariatePoints;

  SetActivePage(1);
  SetViewPort(0,0,GetMaxX,GetMaxY,True);
  SetTextJustify(1,1);
  SetColor(15);  {White}
  OutTextXY (GetMaxX div 2,char_H,'Ordinary Linear Simple Regression
with One Predictor Variable');

  OutTextXY(12,row1+97,'Ho:');
  GMatWrite(Beta1,4,1,#225'10
',44,row1+88,15,BackgroundKolor,MatForeground);       {***** Get Beta1
*****}

GMatWrite(Alpha,4,2,#224,150,row1+88,15,BackgroundKolor,MatForeground);
    {***** Get Beta1 *****}

GMatWrite(Y,4,1,'Y.Full',33,row1,ObservationColor,BackgroundKolor,MatFor
eground);

GMatWrite(X,4,1,'X.Full',150,row1,ModelSpaceColor,BackgroundKolor,MatFor
eground);

  For i:= 1 downto 0 do begin    {Write to Both Buffers}
    SetActivePage(i);

    GMatWrite(Yred,4,1,'Y.Red
',33,row2,Observation2Color,BackgroundKolor,MatForeground);
    GMatWrite(Xred,4,1,'X.Red
',150,row2,ModelSpaceColor,BackgroundKolor,MatForeground);

    BetaX:=315;

GMatWrite(BhatFull,4,1,#225'.Full',BetaX,row1,15,BackgroundKolor,MatFore
ground);
    SetColor(15); OutTextXY(BetaX-21,row1+9,'^');
    SetColor(LabelColor); OutTextXY(BetaX+29,row1-10,'[X''X]  X''Y');
OutTextXY(BetaX+35,row1-15,'-1');


    BetaX:=440;

GMatWrite(YhatFull,4,1,'Y.Full',BetaX,row1,YhatFullColor,BackgroundKolor
,MatForeground);
    SetColor(15); OutTextXY(BetaX-20,row1+12,'^');
    SetColor(LabelColor); OutTextXY(BetaX+60,row1-10,'X'#225);
                OutTextXY(BetaX+60,row1-14,' ^');


GMatWrite(eF,4,1,'e.F',560,row1,ErrorColor,BackgroundKolor,MatForeground
);
    SetColor(LabelColor); OutTextXY(604,row1-10,'Y - Y');
                OutTextXY(604,row1-15,'    ^');
```

```
  {******** Next Row *********}

    BetaX:=315;
    GMatWrite(BhatRed,4,1,#225'.Red
',BetaX,row2,15,BackgroundKolor,MatForeground);
    SetColor(15); OutTextXY(BetaX-21,row2+5,'^');

    BetaX:=440;
    GMatWrite(YhatRed,4,1,'Y.Red
',BetaX,row2,YhatRedColor,BackgroundKolor,MatForeground);
    SetColor(15); OutTextXY(BetaX-20,row2+12,'^');


GMatWrite(eR,4,1,'e.R',560,row2,ErrReducedColor,BackgroundKolor,MatForeg
round);

    row2a:=round(14.5*char_H);
    OutTextXY(355,row2a,'Dimensions');
    SetTextJustify(0,1);
    OutTextXY(235,round(row2a+1.5*char_H),'Sample Space     =   n   = [
]');
    Str(X.rows,msg); OutTextXY(460,round(row2a+1.5*char_H),msg);
    OutTextXY(235,round(row2a+2.7*char_H),'Estimation Space =   p   = [
]');
    Str(X.cols,msg); OutTextXY(460,round(row2a+2.7*char_H),msg);
    OutTextXY(235,round(row2a+3.9*char_H),'Error Space      = n - p = [
]');
    Str(X.rows-X.cols,msg); OutTextXY(460,round(row2a+3.9*char_H),msg);
    SetTextJustify(1,1);

{*****  Bar Graphs  ******}
    row3:=31*char_H;
    row3h:=80;
    SetFillStyle(1,GetBkColor);
    Bar(round(Xlo*GetMaxX)+1,round((1-Yhi)*GetMaxY),GetMaxX,GetMaxY);
    BarGraphs;

    SetFillStyle(1,LabelColor);
    Bar(430,335,580,345); SetColor(0);
    OutTextXY(505,340,'Press F1 for Help');
    SetColor(15);

    SetLineStyle(2,0,1);                    {Block around spin area}
    MoveTo(round(Xlo*GetMaxX),round((1-Yhi)*GetMaxY)-1);
    LineTo(round(Xhi*GetMaxX)+2,round((1-Yhi)*GetMaxY)-1);
    LineTo(round(Xhi*GetMaxX)+2,round((1-Ylo)*GetMaxY));

Line(round(Xhi*GetMaxX)+2,round((1-Yhi)*GetMaxY)-1,GetMaxX,round((1-Yhi)
*GetMaxY)-1);
  end;
end; {Procedure BivariateRoutine}

Procedure ExitRoutine;

begin
  MoleExit;
  CloseGraph;
  Halt;
end; {Procedure ExitRoutine}

Procedure LeaveProgram;

const
  X1:Integer=159;
```

```
  Y1:Integer=135;
  X2:Integer=479;
  Y2:Integer=190;
  col:Integer=25;

var
  i: integer;
  P: Pointer;
  Size: Word;
  ViewPort: ViewPortType;
  OldStyle: TextSettingsType;
  Answer: Char;

begin
  GetTextSettings(OldStyle);
  SetVisualPage(0);
  SetActivePage(1);
  SetViewPort(0,0,GetMaxX,GetMaxY,True);
  Size := ImageSize(X1,Y1,X2,Y2);
  If Size>MemAvail Then writeln(output,'Size=',Size,'
MemAvail=',MemAvail);
  Mark(P);
  GetMem(P, Size);
  GetImage(X1,Y1,X2,Y2,P^);
  SetViewPort(X1,Y1,X2,Y2,True);
  GetViewSettings(ViewPort);
  SetFillStyle(1,15);
  Bar(0,0,X2-X1,Y2-Y1);              {Clear ViewPort}
  SetFillStyle(1,BackgroundKolor);
  Bar(5,4,X2-X1-5,Y2-Y1-4);
  SetColor(15);
  SetTextJustify(1,1);
  OutTextXY((X2-X1) div 2,(Y2-Y1) div 2,'Ready to Quit?');
  SetColor(15);
  SetVisualPage(1);
  repeat
    Answer:=UpCase(ReadKey);
  until (Answer = 'Y') or (Answer = 'N');
  If Answer = 'Y' Then ExitRoutine;
  SetVisualPage(0);
  SetViewPort(0,0,GetMaxX,GetMaxY,True);
  PutImage(X1,Y1,P^,0);     {0=Copy}
  Release(P);
end;

end.
```

```
unit GraphMat;

interface

uses Crt,Dos,Graph,MathMat,TGlobals;

  Procedure GMatInput(var matrx: matx;
                      cell_length: Integer;
                       dec_places: Integer;
                              mat: String;      {Input Matrix Name}
                              x,y: Integer;     {Location}
                         MatColor: Word;        {Color of Matrix}
                    MatBackground: Word;        {Color of Matrix Background}
                    MatForeground: Word);       {Color of Characters}

  Procedure GMatWrite(var matrx: matx;
                      cell_length: Integer;
                       dec_places: Integer;
                              mat: String;      {Input Matrix Name}
                              x,y: Integer;     {Location}
                         MatColor: Word;        {Color of Matrix}
                    MatBackground: Word;        {Color of Matrix Background}
                    MatForeground: Word);       {Color of Characters}

implementation

var
  i,j: integer;
  TLX,TLY,BLX,BLY,TRX,TRY,BRX,BRY: integer;
  ViewPort: ViewPortType;
  P:Pointer;


(*************************************************************************)
(***  The following routines are used in GMatInput and GMatWrite  ***)
(*************************************************************************)

function MaxOf(a,b:integer): integer;

begin
  if a>b then MaxOf:= a
  else MaxOf:= b;
end;

procedure cursor_to_cell(i,j: integer;     {Move Cursor to Current Cell}
                         x,y: integer;     {Location of Entire Matrix}
                         W,H: integer;     {Width & Height of Char Set}
                         mat: string;      {Name of Matrix}
                         cell_length:integer);  {# of digits per cell}

begin
  case j of
    0    : SetViewPort(X+(W-4)*length(mat),Y+H*i,
                       X+W*(cell_length+length(mat)),Y+H*(1+i)-1,False);
    1    : SetViewPort(Round(X+(W-4)*Length(mat)+15),Y+H*i,

Round(X+(W-4)*Length(mat)+15+cell_length*W),Y+H*(1+i)-1,False);
    2    : SetViewPort(Round(X+(W-4)*Length(mat)+10+40),Y+H*i,
                       Round(X+(W-4)*Length(mat)+10+40+cell_length*W),
                       Y+H*(1+i)-1,False);
    3    : SetViewPort(Round(X+(W-4)*Length(mat)+10+80),Y+H*i,
                       Round(X+(W-4)*Length(mat)+10+80+cell_length*W),
                       Y+H*(1+i)-1,False);
```

```
  else   Writeln(output,'Look at else in Cursor_Cell in GraphMat');

  end; {case j}

  GetViewSettings(ViewPort);
{  With ViewPort Do begin
    Write(Output,'i=',i,'  j=',j,'  X1=',X1,'  Y1=',Y1,
                 '  X2=',X2,'  Y2=',Y2);
    if graphresult = -11 then Write(output,'  error')
                         else Write(output,'  ViewPort Set');
    writeln;
    SetLineStyle(0,0,1);
    Rectangle(0,0,X2-X1,Y2-Y1);end;
    Delay(500);}

end; {cursor_to_cell}


procedure brackets(x,y: integer;     {Location of Entire Matrix}
                   W,H: integer;     {Width & Height of Char Set}
                   mat: string;      {Name of Matrix}
                 matrx: matx;        {Matrix containing data}
           cell_length: integer;     {# of digits per cell}
                   F10: Boolean;     {Press F10 or Not}
          BracketColor: Word;        {Color of Bracket}
        MatBackground: Word);         {Color of Background}
var
  i,j: integer;
  UpperTitle,LowerTitle,mat2: string;
  Upper: Boolean;
  OldColor: Word;
  OldStyle: TextSettingsType;
  Size: Word;

begin
  mat2:='';
  GetTextSettings(OldStyle);       {So we can set them back}
  OldColor:=GetColor;
  SetTextJustify(1,1);             {Center Horizontally & Vertically}
  SetColor(15);

    {The routine below looks for a decimal point and assumes that
characters
   following the decimal point will be placed as a subscript.
   Only 1 level of subscripting is supported.}

  Upper:=True;   {Title starts in normal Text}
  UpperTitle:=''; LowerTitle:='';

  for i:=1 to Length(mat) do
  If Copy(mat,i,1) = '.' then Upper:=Not Upper
    else    Case Upper of
                True  : begin UpperTitle:=UpperTitle+Copy(mat,i,1);
                              LowerTitle:=LowerTitle+' '; end;
                False : begin LowerTitle:=LowerTitle+Copy(mat,i,1);
                              UpperTitle:=UpperTitle+' '; end;
  end;

  If length(mat)>0 then begin mat2:=UpperTitle+'=';
LowerTitle:=LowerTitle+' '; end;
  TLX := Round(X+(W-4)*(Length(mat)+2)); TLY := Y;
  TRX := TLX+W*(matrx.cols*Cell_Length+2+ord(matrx.cols>2)*1); TRY:=Y;
  BLX := TLX; BLY:= Y+round(H*(matrx.rows+1.5));
  BRX := TRX; BRY:= BLY;
```

```
  OutTextXY (x,y+Round(0.5*H*(matrx.rows+1.5)),mat2);
  OutTextXY (x,y+Round(0.5*H*(matrx.rows+1.5))+4,LowerTitle);
  SetLineStyle(0,0,3);  {Solid, No Pattern, Thick}
  SetColor(BracketColor);
  MoveTo(TLX+W,TLY); LineTo(TLX,TLY); LineTo(BLX,BLY);
LineTo(BLX+W,BLY);
  MoveTo(TRX-W,TRY); LineTo(TRX,TRY); LineTo(BRX,BRY);
LineTo(BRX-W,BRY);
  SetColor(15);
  If F10 then begin
    SetViewPort(0,0,GetMaxX,GetMaxY,True);
    Size := ImageSize((BLX+BRX) div 2-W*7, Round(BLY+2.5*H),(BLX+BRX)
div 2+W*7, Round(BLY+3.5*H));
    If Size>MemAvail Then writeln(output,'Size=',Size,'
MemAvail=',MemAvail);
    Mark(P);
    GetMem(P, Size);
    GetImage(MaxOf((BLX+BRX) div 2-W*8,0), Round(BLY+3),(BLX+BRX) div
2+W*8, Round(BLY+5*H),P^);
    SetFillStyle(1,MatBackground);
    Bar((BLX+BRX) div 2-W*8, Round(BLY+3),(BLX+BRX) div 2+W*8,
Round(BLY+5*H));
    OutTextXY ((BLX+BRX) div 2, BLY+3*H, 'F10 to finish'); end
   else begin
    SetFillStyle(1,MatBackground);
    Bar(TLX+2,TLY+2,BRX-2,BRY-2); end;

  SetTextJustify(OldStyle.Horiz, OldStyle.Vert);
  SetColor(OldColor);
end;  {brackets}


procedure SetUp(x,y:integer;
                W,H: integer;     {Width & Height of Char Set}
                mat: string;      {Name of Matrix}
              matrx: matx;        {Matrix containing data}
       cell_length:integer;       {# of digits per cell}
                F10: Boolean);    {Press F10 or Not}


begin
  SetViewPort(0,0,GetMaxX,GetMaxY,True);
  If (y+(2+matrx.rows*H))>GetMaxY then begin
    Writeln(output,'Sorry, cursor position is to near the bottom of the
screen.');
     Writeln(output, 'X=',X,'  (W-4)=',W-4,'
length(mat)=',length(mat),'  cell_length=',
     cell_length,'  matrx.cols=',matrx.cols);
    Writeln(output,'Hit <Enter> to Continue');
    readln;
    closegraph; Halt;
    end
  else
   If Round(X+(W-4)*(Length(mat)+2))+W*(matrx.cols*Cell_Length+2+
     ord(matrx.cols>2)*2)>GetMaxX then begin
     Writeln(output, 'Sorry, cursor position is to near the right edge
of the screen.');
     Writeln(output, 'X=',X,'  (W-4)=',W-4,'
length(mat)=',length(mat),'  cell_length=',
     cell_length,'  matrx.cols=',matrx.cols);
     Writeln(output, 'Hit <Enter> to Continue');
     readln;
     closegraph; halt;
     end;
```

```
end;  {Setup}

procedure cell_write(number:string;
                     dec_places:integer;
                    cell_length:integer);

var
  k,errorcode,dplaces: integer;
  s: string;
  x: real;
  OldStyle: TextSettingsType;


begin
  GetTextSettings(OldStyle);       {So we can set them back}
  SetTextJustify(0,2);             {Center Horizontally & Vertically}
  dplaces:=dec_places;
  val(number,x,errorcode);
  repeat
    str(x:cell_length:dplaces,s);
    if length(s)>cell_length then dec(dplaces);
  until ((length(s)<=cell_length) or (dplaces<0));
  OutTextXY(0,1,s);
  SetTextJustify(OldStyle.Horiz, OldStyle.Vert);

end; {cell_write}


function strg(x:real;
    cell_length:integer;
     dec_places:integer) : string;

var
  s: string;

begin
  str(x:cell_length:dec_places,s);
  strg := s;
end; {strg}


Procedure GMatInput(var matrx: matx;
                    cell_length: Integer;
                     dec_places: Integer;
                            mat: String;      {Input Matrix Name}
                            x,y: Integer;     {Location}
                       MatColor: Word;       {Color of Matrix}
                  MatBackground: Word;        {Color of Matrix Background}
                  MatForeground: Word);       {Color of Characters}

var
  H: Integer;     {Height in Pixels of 1 character}
  W: Integer;     {Width in Pixels of 1 character}
 CM: Integer;     {Center of Matrix (X-wise)}
  ViewPort2: ViewPortType;
  OldStyle2: TextSettingsType;


procedure Cursor_Cell(i,j: integer);

begin
  cursor_to_cell(i,j,x,y,W,H,mat,cell_length);
end; {procedure Cursor_Cell}
```

```
procedure Cell_Rite(number:string);
begin
  Cell_Write(number,dec_places,cell_length);
end;  {Procedure Cell_Rite}

procedure cell_text;            {Set Colors to Highlight Current Cell}

begin

  With ViewPort do begin
    SetFillStyle(1,          {Solid Pattern}
                 7);         {Light Gray is Highlighted Cell Background}
    Bar(0,0,X2-X1,Y2-Y1); end;
  SetColor(15); {White is the Highlighted Cells' Foreground}
end; {cell_text}


procedure normal_text(Cell_Color:Word);         {Reset Colors}

begin
  SetColor(Cell_Color); {Normal Cell Foreground}
  With ViewPort do begin
    SetFillStyle(1,  {Solid Pattern}
                 MatBackground); {Blue is the cell background}
    Bar(0,0,X2-X1,Y2-Y1); end;
end; {normal_text}



procedure input_number(var matrx: matx);   {input matrix}

const
  left_arrow = 75;     {0 is returned first for arrow keys!}
  right_arrow = 77;
  up_arrow = 72;
  down_arrow = 80;
  backspace = 8;
  return = 13;
  escape = 27;
  plus = 43;
  minus = 45;
  F10 = 68;
  decimal = 46;

var
  key,key2 : char;
  k,decimals,exit_cell,exit_procedure,errorcode  : Integer;
  number : string;      {input string}


procedure exec_return (var number_string: string;
                       var value: Real;
                       var exit_cell: integer);
var
  k:  integer;
begin
  val(number,value,errorcode);
  exit_cell := 1;        {input is complete}
  If errorcode <> 0 then writeln(output,'error in exec_return
procedure');
  Cursor_Cell(i,j);
  normal_text(LabelColor);
  Cell_Rite(number);
end;  {exec_return}
```

```
procedure too_long;

var
  ViewOld: ViewPortType;
  By,Rx,Lx: Integer;
  msg,cs: String;
  OldStyle: TextSettingsType;

begin
  GetTextSettings(OldStyle);      {So we can set them back}
  GetViewSettings(ViewOld);
  SetViewPort(0,0,GetMaxX,GetMaxY,True);
  By:= Y+round(H*(matrx.rows+1.5));
  Lx := Round(X+(W-4)*(Length(mat)+2));
  Rx := Round(Lx+W*(matrx.cols*Cell_Length+5));

  SetFillStyle(1,          {Solid Pattern}
               MatBackground);

Bar(Round((Lx+Rx)/2-12*W),Trunc(By+3.5*H),Round((Lx+Rx)/2+12*W),Round(By
+2.5*H));
  SetTextJustify(1,1);
  Str(Cell_Length,cs);
  msg:='to '+cs+' digits';

  OutTextXY ((Lx+Rx) div 2,By+H,'Input Limited');
  OutTextXY ((Lx+Rx) div 2,By+2*H,msg);
  OutTextXY ((Lx+Rx) div 2, By+3*H,'(Press RETURN)');
  readln;
  SetFillStyle(1,          {Solid Pattern}
               MatBackground);

Bar(Round((Lx+Rx)/2-7*W),Trunc(By+0.5*H),Round((Lx+Rx)/2+7*W),Round(By+3
.5*H));
  OutTextXY ((LX+RX) div 2, BY+3*H, 'F10 to finish');
  SetTextJustify(OldStyle.Horiz, OldStyle.Vert);

  With ViewOld do SetViewPort(X1,Y1,X2,Y2,False);
end;  {too_long}

procedure char_ok(key:char;
                  var number: string);
var
  x,y,i: integer;
  OldStyle: TextSettingsType;

begin
  GetTextSettings(OldStyle);      {So we can set them back}
  SetTextJustify(0,2);            {Center Horizontally & Vertically}

  number := number + key;
  If length(number) = 1 then       {Blank out old number}
  With ViewPort do begin
    SetFillStyle(1,                {Solid Pattern}
                 MatBackground);   {Highlighted Cell Background}
    Bar(0,0,X2-X1,Y2-Y1); end;

  OutTextXY(GetX,1,key); MoveRel(W,0);
  With OldStyle do          {Reset TextSettings}
   begin
     SetTextJustify(Horiz, Vert);
     SetTextStyle(Font, Direction, CharSize);
   end;
```

```
end;  {char_ok}

function strg(x:real) : string;

var
  s: string;

begin
  str(x:cell_length:dec_places,s);
  strg := s;
end; {strg}

procedure WrapAround;

begin
  if i > matrx.rows  then i := 1;
  if i < 1 then i := matrx.rows ;
  if j > matrx.cols  then j := 1;
  if j < 1 then j := matrx.cols ;
end;  {WrapAround}

procedure ReWrit(i,j: Integer;
              number: string);

begin
   Cursor_Cell(i,j);
   normal_text(LabelColor);
   If length(number) = 0 then Cell_Rite(strg(matrx.data[i,j]))
   else Cell_Rite(number);
end;  {ReWrit}


procedure initialize;

var
  i,j:  integer;
  number: string;

begin
  number := '';
  for i := 1 to matrx.rows  do begin
    for j := 1 to matrx.cols  do rewrit(i,j,number);
    end;  {for i begin}
end;  {Initialize}

(*************************************)
(****   Main Procedure Input_Number ****)
(*************************************)

begin
  Assign(output,'prn');
  Rewrite(output);
  Initialize;
  i := 1;
  j := matrx.cols;
  exit_procedure := 0;
  repeat
    number := '';
    exit_cell := 0;
    decimals := 0;
    Cursor_Cell(i,j);
    cell_text;
    Cell_Rite(strg(matrx.data[i,j]));
    Cursor_Cell(i,j);
```

```
    repeat
      key := readkey;
      if Ord(key) = 0 then key2 := readkey;     {Read Extended Key Code}
  {    Writeln(output,'key 1=',key,'  key2=',key2);    }
      case Ord(key) of
        48..57      :  If length(number) < Cell_Length then
char_ok(key,number)
      { Numbers }      else too_long;

        return      :  If length(number) > 0 then begin
                         exec_return(number,matrx.data[i,j],exit_cell);
                         If i = matrx.rows  then begin
                           i := 1;                {was on bottom row}
                           j := j+1;
                           end  {if i begin}
                         else i := i+1;
                         end {if length begin}
                       else begin
                         rewrit(i,j,number);
                         i := i+1;
                         exit_cell := 1;
                         end; {else begin}

        decimal     :  If decimals = 0 then
                         If length(number) < Cell_Length then begin
                           char_ok(key,number);
                           decimals := 1;
                           end {begin}
                         else too_long;

          0         :   begin
                          case Ord(key2) of
                            left_arrow  :  begin
                                             If length(number)>0 then

exec_return(number,matrx.data[i,j],exit_cell)
                                             else rewrit(i,j,number);
                                             j := j-1;
                                           end;
                            right_arrow :  begin
                                             If length(number)>0 then

exec_return(number,matrx.data[i,j],exit_cell)
                                             else rewrit(i,j,number);
                                             j := j+1;
                                           end;
                            up_arrow    :  begin
                                             If length(number)>0 then

exec_return(number,matrx.data[i,j],exit_cell)
                                             else rewrit(i,j,number);
                                             i := i-1;
                                           end;
                            down_arrow  :  begin
                                             If length(number)>0 then

exec_return(number,matrx.data[i,j],exit_cell)
                                             else rewrit(i,j,number);
                                             i := i+1;
                                           end;
                            F10         :  begin
                                             If length(number) > 0 then

exec_return(number,matrx.data[i,j],exit_cell)
```

```
                                        else rewrit(i,j,number);
                                        exit_procedure := 1;
                                      end;
                            end;  {case Ord(key2)}
                            exit_cell := 1;
                          end;  {begin}
        backspace   :  If length(number)>0 then begin
                         If number[length(number)]='.' then decimals:=0;
                         number := Copy (number,1,(length(number)-1));
                         SetFillStyle(1,          {Solid Pattern}
                          7);         {Light Gray is Highlighted Cell
Background}
                         Bar(GetX-W,GetY+H-1,GetX,GetY);
                         MoveRel(-W,0);
                       end;

        escape      :  begin
                         number := '';
                         With ViewPort do begin
                           SetFillStyle(1,    {Solid Pattern}
                                        7);   {Light Gray is Highlighted
Cell Background}
                           Bar(0,0,X2-X1,Y2-Y1); end;
                         Cursor_Cell(i,j);
                       end;

        plus        :  If length(number) = 0 then char_ok(key,number);

        minus       :  If length(number) = 0 then char_ok(key,number);


        else        ;    {Do Nothing if other keys are pressed}

      end;  {case Ord(key)}
      WrapAround;
    until exit_cell = 1;
  until exit_procedure =1;
  SetViewPort(0,0,GetMaxX,GetMaxY,True);
  PutImage(MaxOf((BLX+BRX) div 2-W*8,0), Round(BLY+3),P^,0);    {0=Copy}
  Release(P);
end;  {input_number}


begin
  H:= TextHeight('H');
  W:= TextWidth('W');
  CM:= X+(W*(Length(mat)+3)+W*(matrx.cols*Cell_Length+1) div 2);
  GetTextSettings(OldStyle2);
  GetViewSettings(ViewPort2);

  SetUp(x,y,W,H,mat,matrx,cell_length,True);
  brackets(x,y,W,H,mat,matrx,cell_length,True,MatColor,MatBackground);
  input_number(matrx);
  With ViewPort2 do SetViewPort(X1,Y1,X2,Y2,Clip);
  With OldStyle2 do
    begin
      SetTextJustify(Horiz, Vert);
      SetTextStyle(Font, Direction, CharSize);
    end;


end;  {M_Input}
```

```
Procedure GMatWrite(var matrx: matx;
                    cell_length: Integer;
                     dec_places: Integer;
                            mat: String;      {Input Matrix Name}
                            x,y: Integer;     {Location}
                       MatColor: Word;        {Color of Matrix}
                  MatBackground: Word;        {Color of Matrix Background}
                  MatForeground: Word);       {Color of Characters}

var
  H: Integer;     {Height in Pixels of 1 character}
  W: Integer;     {Width in Pixels of 1 character}
  ViewPort: ViewPortType;
  OldStyle: TextSettingsType;


begin
  GetTextSettings(OldStyle);
  GetViewSettings(ViewPort);
  W:=TextWidth('W');
  H:=TextHeight('H');
  SetUp(x,y,W,H,mat,matrx,cell_length,False);
  brackets(x,y,W,H,mat,matrx,cell_length,False,MatColor,MatBackground);
  SetColor(MatForeGround);
  for i:=1 to matrx.rows do for j:=1 to matrx.cols do begin
    cursor_to_cell(i,j,x,y,W,H,mat,cell_length);
    Cell_Write (strg(matrx.data[i,j],cell_length,dec_places),
                dec_places,cell_length);
    end;
  With ViewPort do SetViewPort(X1,Y1,X2,Y2,Clip);
  With OldStyle do
    begin
      SetTextJustify(Horiz, Vert);
      SetTextStyle(Font, Direction, CharSize);
    end;


end; {procedure GMatWrite}

end.  {unit}
```

```
unit MathMat;

interface
uses Crt,Dos,Graph;

  Const
    np=5;      {Matrix can be up to 5 x 5, modify np for larger matrics}

  Type
    RealArrayNPbyNP = ARRAY[1..np,1..np] of real;
    matx = Record Data: RealArrayNPbyNP;
                  Rows,Cols: Integer; end;

  Procedure MatInvert (var b,y: matx);     {y=invert of matrix b}

  Procedure MatMult (var a,b,c: matx);     {C=A*B}

  Procedure MatAdd (var a,b,c: matx);      {C=A+B}

  Procedure MatSub (var a,b,c: matx);      {C=A-B}

  Procedure Mat_k_Mult (var a,c: matx;     {C=k*A}
                            k: Real);      {k is a scalar}

  Procedure Mat_Zero (var a: matx);

  Procedure Mat_Transpose (var a,b: matx);  {b is transpose of a}

  Procedure MatWrite (var a: matx);         {Writes an rowxcol matrix}

  Procedure MatInput(var matrx: matx;
               cell_length: Integer;     {Width of each cell}
                dec_places: Integer;     {Number of Decimal Places per
Cell}
                       mat: String;      {Input Matrix Name}
                     x,y: Integer);      {Location}

  Procedure MatAugment(var augmat,mat1,mat2: matx);

implementation

  Type
    RealArrayNP = ARRAY [1..np] of real;
    IntegerArrayNP = ARRAY [1..np] of integer;

var
  i,j: integer;

Procedure ludcmp (var a:     RealArrayNPbyNP;
                      n:     integer;
                  var indx:  IntegerArrayNP;
                  var d:     real);

{Inversion Routine primarily based on routines from the book:
 Numerical Recipes in Pascal by William H. Press and others,
 Published by the Press Syndicate of the University of Cambridge,
 New York, 1989.
 Pages 42-46}

Const
  tiny = 1.0e-20;

Var
  k,j,imax,i:  integer;
```

```
  sum,dum,big:  real;
  vv: ^RealArrayNP;

Begin
  new(vv);
  d:=1.0;
  For i:=1 to n do begin
    big := 0.0;
    for j := 1 to n do
      if abs(a[i,j])>big then big :=abs(a[i,j]);
    if big = 0.0 then begin
      writeln ('pause in LUDCMP - singular matrix');
      readln
    end;
    vv^[i] := 1.0/big
  end;
  for j := 1 to n do begin
    for i := 1 to j-1 do begin
      sum :=a[i,j];
      for k := 1 to i-1 do
        sum := sum-a[i,k]*a[k,j];
      a[i,j] := sum
    end;
    big := 0.0;
    for i := j to n do begin
      sum := a[i,j];
      for k := 1 to j-1 do
        sum := sum-a[i,k]*a[k,j];
      a[i,j] := sum;
      dum := vv^[i]*abs(sum);
      if dum >= big then begin
        big := dum;
        imax :=i
      end
    end;
    if j <> imax then begin
      for k :=1 to n do begin
        dum := a[imax,k];
        a[imax,k] := a[j,k];
        a[j,k] := dum
      end;
      d := -d;
      vv^[imax] := dum
    end;
    indx[j] := imax;
    if a[j,j] = 0.0 then a[j,j] := tiny;
    if j<> n then begin
      dum := 1.0/a[j,j];
      for i := j+1 to n do
        a[i,j]:= a[i,j]*dum
    end
  end;
  dispose(vv)
end;

Procedure lubksb (var a: RealArrayNPbyNP;
                      n: integer;
              var indx: IntegerArrayNP;
                 var b: RealArrayNP);

{Inversion Routine primarily based on routines from the book:
 Numerical Recipes in Pascal by William H. Press and others,
 Published by the Press Syndicate of the University of Cambridge,
 New York, 1989.
```

```
 Pages 42-46}

var
  j,ip,ii,i: integer;
  sum: real;

begin
  ii:=0;

  for i :=1 to n do begin
    ip := indx[i];
    sum := b[ip];
    b[ip] := b[i];
    if ii <> 0 then
      for j := ii to i-1 do
        sum := sum-a[i,j]*b[j]
    else if sum <> 0.0 then
      ii := i;
    b[i] := sum;
  end;
  for i := n downto 1 do begin
    sum := b[i];
    for j:= i+1 to n do
     sum := sum-a[i,j]*b[j];
    b[i] := sum/a[i,i];
  end;
end;

Procedure MatInvert (var b,y: matx);

{Inversion Routine primarily based on routines from the book:
 Numerical Recipes in Pascal by William H. Press and others,
 Published by the Press Syndicate of the University of Cambridge,
 New York, 1989.
 Pages 42-46}

var
  a:    RealArrayNPbyNP;
  i,j:  integer;
  col:  RealArrayNP;
  indx: IntegerArrayNP;
  d:    Real;

begin
  a := b.data;
  ludcmp(a,b.rows,indx,d);
  for j:= 1 to b.rows do begin
    for i:= 1 to b.rows do col[i] :=0.0;
    col[j] := 1.0;
    lubksb(a,b.rows,indx,col);
    for i := 1 to b.rows do y.data[i,j] := col[i]
  end;
  y.rows:=b.rows; y.cols:=b.cols;
end;

Procedure MatMult (var a,b,c: matx);       {C=A*B}

var
  row,col,i: Integer;
  sum: Real;

begin
  if a.cols  = b.rows then begin
    for col := 1 to b.cols do begin
```

```
      for row := 1 to a.rows do begin
        sum := 0;
        for i := 1 to b.rows do
          sum := sum + a.data[row,i]*b.data[i,col];
        c.data[row,col] := sum;
      end;
    end;
  c.rows := a.rows;
  c.cols := b.cols;
  end
  else begin
    writeln ('The number of columns of the first matrix must equal
the');
    writeln ('number of rows of the second matrix in order to
multiply');
    writeln ('matrices.')
   end;
end;


Procedure MatWrite (var a: matx);

var
  i,j: Integer;

begin
  writeln;
  for i := 1 to a.rows do begin
    for j := 1 to a.cols do
      write (a.data[i,j]:5,'  ');
    writeln (' ');
  end;
end;

Procedure MatAdd (var a,b,c: matx);     {C=A+B}

var
  i,j: Integer;

begin
  for i := 1 to a.rows do begin
    for j := 1 to a.cols do
      c.data[i,j] := a.data[i,j] + b.data[i,j];
  end;
  c.rows := a.rows;
  c.cols := a.cols;
end;


Procedure MatSub (var a,b,c: matx);      {C=A-B}

var
  i,j: Integer;

begin
  for i := 1 to a.rows do begin
    for j := 1 to a.cols do
      c.data[i,j] := a.data[i,j] - b.data[i,j];
  end;
  c.rows := a.rows;
  c.cols := a.cols;
end;
```

```
Procedure Mat_k_Mult (var a,c: matx;       {C=k*A}
                          k: Real);        {k is a scalar}

var
  i,j: Integer;

begin
  for i := 1 to a.rows do begin
    for j := 1 to a.cols do
      c.data[i,j] := k*a.data[i,j];
  end;
  c.rows := a.rows;
  c.cols := a.cols;
end;

Procedure Mat_Zero (var a: matx);

var
  m,n:  Integer;

begin
  for m := 1 to a.rows do
    for n := 1 to a.cols do
      a.data[m,n] := 0;
end;

Procedure Mat_Transpose (var a,b: matx);

var
  i,j:  Integer;

begin
  for i := 1 to a.rows do begin
    for j := 1 to a.cols do
      b.data[j,i] := a.data[i,j];
    end;  {for i begin}
  b.rows := a.cols;
  b.cols := a.rows;
end; {Mat_Transpose}



Procedure MatInput(var matrx: matx;
                cell_length: Integer;
                 dec_places: Integer;
                       mat: String;      {Input Matrix Name}
                       x,y: Integer);    {Location}

procedure cursor_to_cell(i,j: integer);    {Move Cursor to Current Cell}

begin
  case j of
    0    : GotoXY(length(mat)+3,i+1);
    1    : GotoXY(length(mat)+5,i+1);
  else    GotoXY(length(mat)+j*cell_length,i+1)
  end; {case j}
end; {cursor_to_cell}


procedure brackets;

const
  open_top_bracket = chr(218);          {ASCII Codes}
```

```
  open_bottom_bracket = chr(192);
  vertical_bar = chr(179);
  close_top_bracket = chr(191);
  close_bottom_bracket = chr(217);

var
  i,j: integer;

begin
  GotoXY(1,trunc((matrx.rows+3)/2));
  Write (mat,'=');
  for i := 0 to matrx.rows+1 do begin
    j := 0;
    cursor_to_cell(i,j);
    If i = 0 then write(open_top_bracket)
    else If i = matrx.rows+1 then write(open_bottom_bracket)
         else write(vertical_bar);

    j := matrx.cols+1;
    cursor_to_cell(i,j);
    If i = 0 then write(close_top_bracket)
    else If i = matrx.rows+1 then write(close_bottom_bracket)
         else write(vertical_bar);

  end;  {loop i}
  GotoXY(2,matrx.rows +4);
  Write('Press F10 when finished');
end;  {brackets}


procedure SetUp(x,y:integer);   {Location of window}

var
  width: integer;

begin
  If (y+matrx.rows +2)>24 then begin
    writeln ('Sorry, cursor position is to near the bottom of the
screen.');
    writeln ('x=',x,'  y=',y);
    writeln ('Hit <Enter> to Continue');
    readln;
    end
  else
   If (x+cell_length*(matrx.cols+1)+3+length(mat))>80 then begin
     writeln ('Sorry, cursor position is to near the right edge of the
screen.');
     writeln ('x=',x,'  y=',y);
     writeln ('Hit <Enter> to Continue');
     readln;
     end;
  Width := cell_length*(matrx.cols+1)+length(mat)+3;
  If Width < 28 then Width := 28;
  Window(x,y,x+Width,y+matrx.rows +3);
  TextBackground(Blue);
  ClrScr;
  brackets;
end;  {Setup}


procedure cell_text;           {Set Colors to Highlight Current Cell}

begin
```

```
  TextBackground(LightGray);
  TextColor(Black);
end; {cell_text}


procedure normal_text;          {Reset Colors}

begin
  TextBackground(Blue);
  TextColor(LightGray);
end; {normal_text}



procedure input_number(var matrx: matx);    {input matrix}

const
  left_arrow = 75;      {0 is returned first for arrow keys!}
  right_arrow = 77;
  up_arrow = 72;
  down_arrow = 80;
  backspace = 8;
  return = 13;
  escape = 27;
  plus = 43;
  minus = 45;
  F10 = 68;
  decimal = 46;

var
  key,key2 : char;
  k,dec,exit_cell,exit_procedure,errorcode  : Integer;
  number : string;      {input string}


procedure cell_write(number:string);

var
  k,errorcode: integer;
  s: string;
  x: real;

begin
  val(number,x,errorcode);
  str(x:cell_length:dec_places,s);
  write(s);
end; {cell_write}


procedure exec_return (var number_string: string;
                       var value: Real;
                       var exit_cell: integer);
var
  k:  integer;
begin
  val(number,value,errorcode);
  exit_cell := 1;         {input is complete}
  If errorcode <> 0 then writeln ('error in exec_return procedure');
  Cursor_to_Cell(i,j);
  normal_text;
  cell_write(number);
end;  {exec_return}
```

```
procedure too_long;

var
  h,m,s,s100,s_delay,s_count: Word;

begin
  GotoXY(2,matrx.rows +3);
  Write('Input Limited to ',Cell_Length,' digits');
  GetTime(h,m,s,s100);
  s_delay := s+3;
  GotoXY(2,matrx.rows +3);
  repeat
    GetTime(h,m,s,s100);
  until s >= s_delay;
  TextBackground(Blue);
  Write('                              ');
end;  {too_long}

procedure char_ok(key:char;
                  var number: string);
var
  x,y,i: integer;

begin
  write(key);
  number := number + key;
  If length(number) = 1 then begin
    x := WhereX;
    y := WhereY;
    For i := 2 to Cell_Length do write(' ');
    GotoXY(x,y);
  end; {if}
end;  {char_ok}

function strg(x:real) : string;

var
  s: string;

begin
  str(x:cell_length:dec_places,s);
  strg := s;
end; {strg}

procedure WrapAround;

begin
  if i > matrx.rows  then i := 1;
  if i < 1 then i := matrx.rows ;
  if j > matrx.cols  then j := 1;
  if j < 1 then j := matrx.cols ;
end;  {WrapAround}

procedure ReWrit(i,j: Integer);

begin
   Cursor_to_Cell(i,j);
   normal_text;
   If length(number) = 0 then cell_write(strg(matrx.data[i,j]))
   else cell_write(number);
end;  {ReWrit}


procedure initialize;
```

```
var
  i,j:  integer;
  number: string;

begin
  number := '';
  for i := 1 to matrx.rows  do begin
    for j := 1 to matrx.cols  do ReWrit(i,j);
    end;  {for i begin}
end;  {Initialize}

(*****************************************)
(****   Main Procedure Input_Number ****)
(*****************************************)

begin
  Initialize;
  i := 1;
  j := 1;
  exit_procedure := 0;
  repeat
    number := '';
    exit_cell := 0;
    dec := 0;
    cursor_to_cell(i,j);
    cell_text;
    cell_write(strg(matrx.data[i,j]));
    cursor_to_cell(i,j);
    repeat
      key := readkey;
      if Ord(key) = 0 then key2 := readkey;     {Read Extended Key Code}
      case Ord(key) of
        48..57      :  If length(number) < Cell_Length then
char_ok(key,number)
      { Numbers }      else too_long;

        return      :  If length(number) > 0 then begin
                         exec_return(number,matrx.data[i,j],exit_cell);
                         If i = matrx.rows  then begin
                           i := 1;                {was on bottom row}
                           j := j+1;
                           end  {if i begin}
                         else i := i+1;
                         end {if length begin}
                       else begin
                         ReWrit(i,j);
                         i := i+1;
                         exit_cell := 1;
                         end; {else begin}

        decimal     :  If dec = 0 then
                         If length(number) < Cell_Length then begin
                           char_ok(key,number);
                           dec := 1;
                           end {begin}
                         else too_long;

           0        :   begin
                          case Ord(key2) of
                            left_arrow  :  begin
                                             If length(number)>0 then

exec_return(number,matrx.data[i,j],exit_cell)
                                             else ReWrit(i,j);
```

```
                                        j := j-1;
                                      end;
                          right_arrow : begin
                                        If length(number)>0 then
exec_return(number,matrx.data[i,j],exit_cell)
                                        else ReWrit(i,j);
                                        j := j+1;
                                      end;
                          up_arrow    : begin
                                        If length(number)>0 then
exec_return(number,matrx.data[i,j],exit_cell)
                                        else ReWrit(i,j);
                                        i := i-1;
                                      end;
                          down_arrow  : begin
                                        If length(number)>0 then
exec_return(number,matrx.data[i,j],exit_cell)
                                        else ReWrit(i,j);
                                        i := i+1;
                                      end;
                          F10         : begin
                                        If length(number) > 0 then
exec_return(number,matrx.data[i,j],exit_cell)
                                        else ReWrit(i,j);
                                        exit_procedure := 1;
                                        GotoXY(2,matrx.rows +4);
                                        Write('
 ');
                                      end;
                         end;  {case Ord(key2)}
                         exit_cell := 1;
                       end;  {begin}
        backspace    : begin
                         number := Copy (number,1,(length(number)-1));
                         GotoXY(WhereX-1,WhereY);
                         write (' ');
                         GotoXY(WhereX-1,WhereY);
                       end;

        escape       : begin
                         number := '';
                         Cursor_to_Cell(i,j);
                         for k := 1 to Cell_Length do write (' ');
                         Cursor_to_Cell(i,j);
                       end;

        plus         : If length(number) = 0 then char_ok(key,number);

        minus        : If length(number) = 0 then char_ok(key,number);


        else         ;    {Do Nothing if other keys are pressed}

      end;  {case Ord(key)}
      WrapAround;
    until exit_cell = 1;
    normal_text;
  until exit_procedure =1;
end;  {input_number}
```

```
begin
  Setup(x,y);
  input_number(matrx);
  Window(1,1,80,25);
end;  {M_Input}

Procedure MatAugment(var augmat,mat1,mat2: matx);

var
  i,j:  Integer;

begin
  for i := 1 to mat1.rows do begin
    for j := 1 to (mat1.cols+mat2.cols) do
    if j <= mat1.cols then augmat.data[i,j] := mat1.data[i,j]
    else augmat.data[i,j] := mat2.data[i,j-mat1.cols];
  end; {for i}
  augmat.rows := mat1.rows;
  augmat.cols := mat1.cols+mat2.cols;
end; {MatAugment}

end.  {unit}
```

```
Unit mole;

interface
uses TGlobals,Graph,Dos;

Procedure MoleInit;
Procedure MoleVideoInit(Background: Integer);
Procedure MoleClip(Xlo,Ylo,Xhi,Yhi: Real);
Procedure MolePlot(Background: Integer);
Procedure MolePlusMinus;
Procedure MoleLeftRight;
Procedure MoleUpDown;
Procedure MoleExit;
Procedure MoleTranslateHorizontal;

implementation

Procedure MoleInit;

Begin
(*-------------------------------------------------------------------------
------*)
(* Call BeginAcroMole to initialize AcroMole, and to ensure that
AcroMole can *)
(* run on this system.
    *)
(*-------------------------------------------------------------------------
------*)
BeginAcroMoleVar.Version:=1; BeginAcroMoleVar.Revision:=0;
BeginAcroMoleVar.AcroMoleSize:=SizeOf(AcroMole);
BeginAcroMoleVar.AcroMolePointer:=Addr(AcroMole);
BeginAcroMole(BeginAcroMoleVar);
If BeginAcroMoleVar.ReturnCode<>AMOkay Then
   If BeginAcroMoleVar.ReturnCode=AMInvalidDOSVersion Then Begin
      Writeln('AcroMol'#130' requires DOS 2.0 or greater.'); Halt(1);
End
   Else Begin Writeln('Internal error.'); Halt(2); End;
(*-------------------------------------------------------------------------
------*)
(* Get AcroMole's suggested video mode, and ensure that this system
supports  *)
(* at least one video mode.
    *)
(*-------------------------------------------------------------------------
------*)
DetectVideoModeVar.VideoMode:=6;
DetectVideoModeVar.MaximumBuffer:=1;
AcroMole.DetectVideoMode(DetectVideoModeVar);

If DetectVideoModeVar.ReturnCode<>AMOkay Then Begin
 Writeln('This program requires a 256K EGA video adapter');
   AcroMole.EndAcroMole(EndAcroMoleVar); Halt(3); End;
(*-------------------------------------------------------------------------
------*)
(* Initialize various fields in the data blocks for the AcroMole
subroutines. *)
(*-------------------------------------------------------------------------
------*)
GetVideoModeInfoVar.VideoMode:=6;
CalculateScaleFactorsVar.SizeX:=65535;      (* Initialize the screen
size to  *)
CalculateScaleFactorsVar.SizeY:=49152;      (* the standard 4:3 ratio.
    *)
```

```
CalculateScaleFactorsVar.ScaleFactorLo:=0;    (* Magnify the picture by
a     *)
CalculateScaleFactorsVar.ScaleFactorHi:=120;  (* factor of 120.
    *)
(*-------------------------------------------------------------------------
------*)
(* Initialize some miscellaneous variables.
    *)
(*-------------------------------------------------------------------------
------*)
Radius:=0.62;                          {Initial Position}
Set3DCameraVar.Perspective:=1000;      (* Enable perspective for camera.
*)
(*-------------------------------------------------------------------------
------*)
(* This is the top of the main loop.
    *)
(*-------------------------------------------------------------------------
------*)
End;

Procedure MoleVideoInit(Background: Integer);

var
  I,J:  Integer;
begin
Assign(Output, 'prn');
Rewrite(Output);
      AcroMole.GetVideoModeInfo(GetVideoModeInfoVar);

(*-------------------------------------------------------------------------
*)
      (* Calculate the scale factors for Set3DCamera so that the object
    *)
      (* will be the same size independent of the video mode's
resolution.    *)

(*-------------------------------------------------------------------------
*)
      CalculateScaleFactorsVar.ResolutionX:=

GetVideoModeInfoVar.MaximumScreenX-GetVideoModeInfoVar.MinimumScreenX+1;
      CalculateScaleFactorsVar.ResolutionY:=

GetVideoModeInfoVar.MaximumScreenY-GetVideoModeInfoVar.MinimumScreenY+1;
      AcroMole.CalculateScaleFactors(CalculateScaleFactorsVar);

Set3DCameraVar.ScaleFactorX:=CalculateScaleFactorsVar.ScaleFactorX;

Set3DCameraVar.ScaleFactorY:=CalculateScaleFactorsVar.ScaleFactorY;

(*-------------------------------------------------------------------------
*)
      (* Set the video mode.
    *)

(*-------------------------------------------------------------------------
*)
      SetVideoModeVar.VideoMode:=GetVideoModeInfoVar.VideoMode;
      SetVideoModeVar.Reset:=0;    {1 resets hardware, 0 doesn't}
      AcroMole.SetVideoMode(SetVideoModeVar);
      Acromole.GetVideoModeInfo(GetVideoModeInfoVar);
      ScreenWidth:=Round(GetVideoModeInfoVar.MaximumScreenX -
                  GetVideoModeInfoVar.MinimumScreenX);
```

```
      ScreenHeight:=Round(GetVideoModeInfoVar.MaximumScreenY -
                          GetVideoModeInfoVar.MinimumScreenY);

(*-------------------------------------------------------------------------
*)
      (* Set the clipping window to the screen.
    *)

(*-------------------------------------------------------------------------
*)
        SetWindowVar.MinimumFilmX:=-Round(ScreenWidth/2);
        SetWindowVar.MinimumFilmY:=-Round(ScreenHeight/2);
        SetWindowVar.MaximumFilmX:=Round(ScreenWidth/2);
        SetWindowVar.MaximumFilmY:=Round(ScreenHeight/2);
        SetWindowVar.MinimumScreenX:=GetVideoModeInfoVar.MinimumScreenX;
        SetWindowVar.MinimumScreenY:=GetVideoModeInfoVar.MinimumScreenY;
      AcroMole.SetWindow(SetWindowVar);


(*-------------------------------------------------------------------------
*)
      (* Clear the screen.  If the video mode supports two or more
buffers,   *)
      (* so that we can double buffer, clear the first two buffers.
    *)

(*-------------------------------------------------------------------------
*)

      DrawRectangleVar.ScreenX1:=GetVideoModeInfoVar.MinimumScreenX;
      DrawRectangleVar.ScreenY1:=GetVideoModeInfoVar.MinimumScreenY;
      DrawRectangleVar.ScreenX2:=GetVideoModeInfoVar.MaximumScreenX;
      DrawRectangleVar.ScreenY2:=GetVideoModeInfoVar.MaximumScreenY;

      DrawRectangleVar.Color:=Background; ToErase:=0;
      If GetVideoModeInfoVar.MaximumBuffer=0 Then ToDraw:=1
      Else Begin ToDraw:=2;
         AcroMole.DrawRectangle(DrawRectangleVar);
         SetDrawingBufferVar.Buffer:=1;
         SetDisplayedBufferVar.Buffer:=0;
         AcroMole.SetDrawingBuffer(SetDrawingBufferVar);
        End;
      AcroMole.DrawRectangle(DrawRectangleVar);

(*-------------------------------------------------------------------------
*)
      (* Set the number of points and lines on the screen to zero.
    *)

(*-------------------------------------------------------------------------
*)
       For I:=0 To 2 Do Begin Points[I]:=0; Lines[I]:=0; End;
end; {Procedure MoleVideoInit}

Procedure MoleClip(Xlo,Ylo,Xhi,Yhi: Real);

var
  ScreenWidth,ScreenHeight: Integer;

begin
  ScreenWidth:=Round(GetVideoModeInfoVar.MaximumScreenX -
                     GetVideoModeInfoVar.MinimumScreenX);
  ScreenHeight:=Round(GetVideoModeInfoVar.MaximumScreenY -
                      GetVideoModeInfoVar.MinimumScreenY);
```

```
(*-------------------------------------------------------------------------
*)
  (* Set the clipping window to the screen.
*)

(*-------------------------------------------------------------------------
*)
  SetWindowVar.MinimumFilmX:=-Round((Xhi-Xlo)*ScreenWidth/2);
  SetWindowVar.MinimumFilmY:=-Round((Yhi-Ylo)*ScreenHeight/2);
  SetWindowVar.MaximumFilmX:=Round((Xhi-Xlo)*ScreenWidth/2);
  SetWindowVar.MaximumFilmY:=Round((Yhi-Ylo)*ScreenHeight/2);

SetWindowVar.MinimumScreenX:=Round(Xlo*ScreenWidth+GetVideoModeInfoVar.M
inimumScreenX);

SetWindowVar.MinimumScreenY:=Round(Ylo*ScreenHeight+GetVideoModeInfoVar.
MinimumScreenY);

  AcroMole.SetWindow(SetWindowVar);
end;  {Procedure MoleClip}


Procedure MolePlot(Background: Integer);

var
  I,J:  Integer;

Begin

(*-------------------------------------------------------------------------
---*)
   (* Set the camera position. . The camera rotates around the object,
    *)
   (* which makes the paddlewheel look like it is spinning.
    *)

(*-------------------------------------------------------------------------
---*)
   With Set3DCameraVar Do
   Begin
     WorldX:=-Round(Radius*RotMat[0,2]);
     WorldY:=-Round(Radius*RotMat[1,2]);
     WorldZ:=-Round(Radius*RotMat[2,2]);
     DirectionX:=Round(RotMat[0,2]);
     DirectionY:=Round(RotMat[1,2]);
     DirectionZ:=Round(RotMat[2,2]);
     UpX:=Round(RotMat[0,1]);
     UpY:=Round(RotMat[1,1]);
     UpZ:=Round(RotMat[2,1]);
   end;

   AcroMole.Set3DCamera(Set3DCameraVar);

(*-------------------------------------------------------------------------
---*)
   (* Transform the endpoints of the lines.
    *)

(*-------------------------------------------------------------------------
---*)
   For I:=1 To NumberOfEndpoints Do

AcroMole.Transform3DEndpoint(WorldEndpoint[I-1].Transform3DEndpointVar);
```

```
(*-------------------------------------------------------------------------
---*)
   (* Clip any lines that we cannot immediately accept or reject.  If
the     *)
   (* OR of the clipping bits is zero, then we can immediately accept
the     *)
   (* line, and if the AND is non-zero, then we can immediately reject
the    *)
   (* line.
    *)

(*-------------------------------------------------------------------------
---*)
   Lines[ToDraw]:=0;
   For I:=1 To NumberOfLines Do
      With WorldLine[I-1], ScreenLine[ToDraw,Lines[ToDraw]] Do

       If Layer and LayerScreen <> 0 then

         If
(WorldEndpoint[Endpoint1].Transform3DEndpointVar.ClippingBits Or

WorldEndpoint[Endpoint2].Transform3DEndpointVar.ClippingBits)=0
         Then Begin
            With WorldEndpoint[Endpoint1] Do Begin
               DrawLineVar.ScreenX1:=Transform3DEndpointVar.ScreenX;
               DrawLineVar.ScreenY1:=Transform3DEndpointVar.ScreenY;
End;
            With WorldEndpoint[Endpoint2] Do Begin
               DrawLineVar.ScreenX2:=Transform3DEndpointVar.ScreenX;
               DrawLineVar.ScreenY2:=Transform3DEndpointVar.ScreenY;
End;
            DrawLineVar.Color:=LineColor; Inc(Lines[ToDraw]); End
         Else If
(WorldEndpoint[Endpoint1].Transform3DEndpointVar.ClippingBits
            And
WorldEndpoint[Endpoint2].Transform3DEndpointVar.ClippingBits)=0
         Then Begin
            With WorldEndpoint[Endpoint1] Do Begin

Clip3DLineVar.ClippingBits1:=Transform3DEndpointVar.ClippingBits;
               Clip3DLineVar.ScreenX1:=Transform3DEndpointVar.ScreenX;
               Clip3DLineVar.ScreenY1:=Transform3DEndpointVar.ScreenY;
               Clip3DLineVar.CameraX1:=Transform3DEndpointVar.CameraX;
               Clip3DLineVar.CameraY1:=Transform3DEndpointVar.CameraY;
               Clip3DLineVar.CameraZ1:=Transform3DEndpointVar.CameraZ;
End;
            With WorldEndpoint[Endpoint2] Do Begin

Clip3DLineVar.ClippingBits2:=Transform3DEndpointVar.ClippingBits;
               Clip3DLineVar.ScreenX2:=Transform3DEndpointVar.ScreenX;
               Clip3DLineVar.ScreenY2:=Transform3DEndpointVar.ScreenY;
               Clip3DLineVar.CameraX2:=Transform3DEndpointVar.CameraX;
               Clip3DLineVar.CameraY2:=Transform3DEndpointVar.CameraY;
               Clip3DLineVar.CameraZ2:=Transform3DEndpointVar.CameraZ;
End;
            AcroMole.Clip3DLine(Clip3DLineVar);
            If Clip3DLineVar.ReturnCode=AMOkay Then Begin
                  DrawLineVar.ScreenX1:=Clip3DLineVar.ScreenX1;
                  DrawLineVar.ScreenY1:=Clip3DLineVar.ScreenY1;
                  DrawLineVar.ScreenX2:=Clip3DLineVar.ScreenX2;
                  DrawLineVar.ScreenY2:=Clip3DLineVar.ScreenY2;
```

```
                    DrawLineVar.Color:=LineColor; Inc(Lines[ToDraw]); End;
End;

(*-------------------------------------------------------------------------
---*)
   (* Draw the points and lines on the screen.
    *)

(*-------------------------------------------------------------------------
---*)
   If GetVideoModeInfoVar.MaximumBuffer>=1 Then Begin

(*-------------------------------------------------------------------------
*)
      (* If we are double buffering, first erase all of the old points
and    *)
      (* lines in the non-visible buffer, then draw all of the new
points and *)
      (* lines, and then switch buffers.
    *)

(*-------------------------------------------------------------------------
*)
      For I:=1 To Lines[ToErase] Do Begin                  (* Erase old
lines. *)
         With ScreenLine[ToErase,I-1] Do Begin
DrawLineVar.Color:=Background;
            AcroMole.DrawLine(DrawLineVar); End; End;
      For I:=1 To Lines[ToDraw] Do                          (* Draw new
lines. *)
         AcroMole.DrawLine(ScreenLine[ToDraw,I-1].DrawLineVar);
      SetDrawingBufferVar.Buffer:=(SetDrawingBufferVar.Buffer+1) Mod 2;
      AcroMole.SetDrawingBuffer(SetDrawingBufferVar);     (* Switch
drawing.  *)
      SetDisplayedBufferVar.Buffer:=(SetDisplayedBufferVar.Buffer+1) Mod
2;
      AcroMole.SetDisplayedBuffer(SetDisplayedBufferVar); (* Switch
displayed.*)
      ToErase:=(ToErase+1) Mod 3; ToDraw:=(ToDraw+1) Mod 3;
      Repeat AcroMole.CheckVerticalRetrace(CheckVerticalRetraceVar);
      Until CheckVerticalRetraceVar.ReturnCode=AMOkay; End

(*-------------------------------------------------------------------------
*)
      (* If we are single buffering, erase and redraw one point and one
line  *)
      (* at a time, so that the screen is never entirely blank.
    *)

(*-------------------------------------------------------------------------
*)
   Else Begin J:=Points[ToErase]; If J>Points[ToDraw] Then
J:=Points[ToDraw];
      J:=Lines[ToErase]; If J>Lines[ToDraw] Then J:=Lines[ToDraw];
      For I:=J+1 To Lines[ToErase] Do
         With ScreenLine[ToErase,I-1] Do Begin
DrawLineVar.Color:=Background;
            AcroMole.DrawLine(DrawLineVar); End;
      For I:=J+1 To Lines[ToDraw] Do
         AcroMole.DrawLine(ScreenLine[ToDraw,I-1].DrawLineVar);
      ToErase:=(ToErase+1) Mod 2; ToDraw:=(ToDraw+1) Mod 2; End;

(*-------------------------------------------------------------------------
---*)
```

```
   (* Synchronize with vertical retrace.
    *)

(*-------------------------------------------------------------------------
---*)
   Repeat AcroMole.CheckVerticalRetrace(CheckVerticalRetraceVar);
   Until CheckVerticalRetraceVar.ReturnCode=AMVerticalRetrace;

(*-------------------------------------------------------------------------
---*)
   (* Get the keyboard status, check the status of the CTRL and BREAK
(or     *)
   (* CTRL and C) keys, and clear the keyboard buffer.
    *)

(*-------------------------------------------------------------------------
---*)
   AcroMole.GetKeyboardStatus(GetKeyboardStatusVar);
   AcroMole.CheckBreak(CheckBreakVar);
   AcroMole.ClearKeyboardBuffer(ClearKeyboardBufferVar);
end; {Procedure MolePlot}

Procedure MolePlusMinus;
var
  i,j: integer;

(*-------------------------------------------------------------------------
---*)
   (* The plus and minus arrow keys move you in and out from the box.
*)
   (* Don't move any closer than 0, nor any further than 32767.
    *)

(*-------------------------------------------------------------------------
---*)

Begin

   If (GetKeyboardStatusVar.Down[4] And 16384)<>0 (* Plus key held down.
      *)
   Then Radius:=Radius-0.005;          (* Move toward Box.    *)
   If (GetKeyboardStatusVar.Down[4] And 1024)<>0 (* Minus key held down.
      *)
   Then Radius:=Radius+0.005;          (* Move away from Box.   *)
   If Radius<0.0000001 Then Radius:=0.0000001; If Radius>1.0 Then
Radius:=1.0;
end;  {Procedure MolePlusMinus}

Procedure MoleLeftRight;

(*-------------------------------------------------------------------------
---*)
   (* The left and right arrow keys change the angle along the X and Z
axis.    *)
   (* Save the new scale factors for Set3DCamera.
    *)

(*-------------------------------------------------------------------------
---*)

var
  TempReal:  Real;

Begin
```

```
    If (GetKeyboardStatusVar.Down[4] And 8192)<>0  {Right arrow held
down}
    Then SinAngle:=SinPosAngle
    else SinAngle:=-SinPosAngle;                    {Left Arrow}
    TempReal:=RotMat[0,0];
    RotMat[0,0]:=CosAngle*RotMat[0,0]-SinAngle*RotMat[0,2];
    RotMat[0,2]:=SinAngle*TempReal+CosAngle*RotMat[0,2];
    TempReal:=RotMat[1,0];
    RotMat[1,0]:=CosAngle*RotMat[1,0]-SinAngle*RotMat[1,2];
    RotMat[1,2]:=SinAngle*TempReal+CosAngle*RotMat[1,2];
    TempReal:=RotMat[2,0];
    RotMat[2,0]:=CosAngle*RotMat[2,0]-SinAngle*RotMat[2,2];
    RotMat[2,2]:=SinAngle*TempReal+CosAngle*RotMat[2,2];
end; {Procedure MoleLeftRight;}


Procedure MoleUpDown;

var
  TempReal:  Real;

Begin
  If (GetKeyboardStatusVar.Down[4] And 256)<>0   {Up arrow held down}
  Then SinAngle:=SinPosAngle
  else SinAngle:=-SinPosAngle;                   {Down Arrow}
  TempReal:=RotMat[0,1];
  RotMat[0,1]:=CosAngle*RotMat[0,1]-SinAngle*RotMat[0,2];
  RotMat[0,2]:=SinAngle*TempReal+CosAngle*RotMat[0,2];
  TempReal:=RotMat[1,1];
  RotMat[1,1]:=CosAngle*RotMat[1,1]-SinAngle*RotMat[1,2];
  RotMat[1,2]:=SinAngle*TempReal+CosAngle*RotMat[1,2];
  TempReal:=RotMat[2,1];
  RotMat[2,1]:=CosAngle*RotMat[2,1]-SinAngle*RotMat[2,2];
  RotMat[2,2]:=SinAngle*TempReal+CosAngle*RotMat[2,2];
End; {Procedure MoleUpDown}

Procedure MoleExit;

(*-------------------------------------------------------------------------
------*)
(* Restore the original video mode, clear extraneous keystrokes from the
    *)
(* keyboard buffer, end AcroMole, and return to DOS.
    *)
(*-------------------------------------------------------------------------
------*)
Begin
  AcroMole.RestoreOriginalVideoMode(RestoreOriginalVideoModeVar);
  AcroMole.ClearKeyboardBuffer(ClearKeyboardBufferVar);
  AcroMole.EndAcroMole(EndAcroMoleVar);
End;


End.
```

Bibliography

1. Bryant, Peter. "Geometry, Statistics, Probability: Variations on a Common Theme," *The American Statistician, 38:* 38-48 (February 1984).

2. Cowley, Geoffrey and others. "Not Just for Nerds," *Newsweek,* 52-54 (9 April 1990).

3. Devore, Jay L. *Probability and Statistics for Engineering and the Sciences* (Second Edition). Monterey CA: Brooks/Cole Publishing Company, 1987.

4. Glaserfeld, Ernst Von. "Learning as a Constructive Activity," *Problems of Representation in the Teaching and Learning of Mathematics,* edited by Claude Janvier. Hillsdale NJ: Lawrence Eribaum Associates, Publishers, 1987.

5. Hansard, Capt Stone W. An Interactive System of Computer Generated Graphic Displays for Motivating Meaningful Learning of Matrix Operations and Concepts of Matrix Algebra. MS Thesis, AFIT/GSM/ENC/90S-12. School of Systems and Logistics, Air Force Institute of Technology (AU), Wright-Patterson AFB OH, September 1990 (AD-A229557).

6. Herr, David G. "On the History of the Use of Geometry in the General Linear Model," *The American Statistician, 34:* 43-47 (February 1980).

7. Levin, Richard I. *Statistics for Management* (Fourth Edition). Englewood Cliffs NJ: Prentice-Hall, Inc., 1987.

8. Margolis, Marvin S. "Perpendicular Projections and Elementary Statistics," *The American Statistician, 33:* 131-135 (August 1979).

9. Novak, Joseph D. and D. Bob Gowin. *Learning How to Learn.* New York: Cambridge University Press, 1984.

10. Saville, D. J. and G. R. Wood. "A Method for Teaching Statistics Using N-Dimensional Geometry," *The American Statistician, 40:* 205-213 (August 1986).

11. Scandura, Joseph M. "Research in Mathematics Education - An Overview and a Perspective," *Research in Mathematics Education,* Edited by Joseph M. Scandura. Washington: National Council of Teachers of Mathematics, 1967.

Vita

Captain Stephen D. Pearce was born on 2 September 1963 in Chattanooga, Tennessee. He graduated from Hixson High School in Chattanooga in 1981. With a four year Air Force Reserve Officer Training Corps scholarship, he attended Auburn University. In 1985 he graduated with a Bachelor of Electrical Engineering degree and was commissioned in the USAF as a Second Lieutenant. His first assignment was to the Armament Division at Eglin AFB, Florida as a configuration manager in the Advanced Medium Range Air-to-Air Missile Joint Systems Program Office. Three years later he was promoted to chief of the configuration management division. As a configuration manager Captain Pearce conducted many Physical Configuration Audits, Functional Configuration Audits, and Engineering Drawing Reviews. In May 1989 he married Melissa Conrad. In May 1990 he entered the Air Force Institute of Technology at Wright Patterson AFB, Ohio where he worked to obtain a Master of Science in Systems Management degree.

Permanent Address:

2211 Belmont Road
Arnoldsville, GA 30619

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	September 1991	Master's Thesis

4. TITLE AND SUBTITLE
AN APPLICATION OF INTERACTIVE COMPUTER GRAPHICS TO THE STUDY OF INFERENTIAL STATISTICS AND THE GENERAL LINEAR MODEL

5. FUNDING NUMBERS

6. AUTHOR(S)
Stephen D. Pearce, Capt, USAF

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Air Force Institute of Technology, WPAFB OH 45433

8. PERFORMING ORGANIZATION REPORT NUMBER
AFIT/GSM/ENC/91S-22

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This research created a learning environment, known as the Pearce Projection Modeling Environment (PPME) which is used as a tool by a teacher and student. The PPME was developed in an effort to create a new approach to the study of the General Linear Model through a constructive and projective, geometric approach. While the geometric approach to the GLM was developed over the past century, it has not been used extensively because of the inherent complexities associated with visualizing vector spaces. With the PPME, visualization is accomplished effortlessly. The PPME is a computer program that allows the student to enter response vectors and other vectors and data associated with the GLM and observe the relationships of those vectors interactively and in three-dimensions. The PPME encourages learning through constructive development by allowing the student to modify the vectors and observe the results of his actions.

To validate the PPME as a learning tool, several data sets were generated and used to study three scenarios: The Sample Mean and Variance, A General Linear Test of the Population Mean, and An Ordinary Least Squares Simple Linear Regression.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Statistics, Learning, Graphics, Vector Spaces, Computer Programs, General Linear Model	169
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

AFIT Control Number AFIT/GSM/ENC/91S-22

AFIT RESEARCH ASSESSMENT

The purpose of this questionnaire is to determine the potential for current and future applications of AFIT thesis research. Please return completed questionnaires to: AFIT/LSC, Wright-Patterson AFB OH 45433-6583.

1. Did this research contribute to a current research project?

   a. Yes b. No

2. Do you believe this research topic is significant enough that it would have been researched (or contracted) by your organization or another agency if AFIT had not researched it?

   a. Yes b. No

3. The benefits of AFIT research can often be expressed by the equivalent value that your agency received by virtue of AFIT performing the research. Please estimate what this research would have cost in terms of manpower and/or dollars if it had been accomplished under contract or if it had been done in-house.

   Man Years _____ $_____

4. Often it is not possible to attach equivalent dollar values to research, although the results of the research may, in fact, be important. Whether or not you were able to establish an equivalent value for this research (3 above), what is your estimate of its significance?

   a. Highly Significant b. Significant c. Slightly Significant d. Of No Significance

5. Comments

Name and Grade _____ Organization _____

Position or Title _____ Address _____